



Grant Agreement No.: 732638
 Call: H2020-ICT-2016-2017
 Topic: ICT-13-2016
 Type of action: RIA



D3.6: Developments for the 3rd cycle

Work package	WP 3
Task	Task 3.1 – 3.5
Due date	31/10/2021
Submission date	17/03/2022
Deliverable lead	Imec
Version	4
Authors	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Albert (Yiu Quan) Su (SU), Dimitris Dechouniotis (NTUA), Costas Papadakis (NTUA), Aris Dadoukis (CERTH), Donatos Stavropoulos (CERTH), Ana Juan Ferrer (ATOS), Roman Sosa Gonzalez (ATOS), Rowshan Jahan Sathi (TUB), Alex Willner (TUB), Lucas Nussbaum (Inria), David Margery (Inria), Cedric Crettaz (MI), Carolina Fernández (i2CAT), Diarmuid Collins (TCD)
Reviewers	Peter Van Daele (imec), Maria Chiara Campodonico (Martel)
Abstract	This deliverable gives an overview of the developments in WP3 during the 3rd cycle. WP2 are normal operations developments (add testbeds, fix bugs, small features, etc). WP3 is focusing on larger new functionality.
Keywords	Developments third cycle, new functionality

DOCUMENT REVISION HISTORY

Version	Date	Description of change	List of contributor(s)
V1	1/10/2021	TOC	Brecht Vermeulen (imec)
V2	22/12/2021	First version	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Albert (Yiu Quan) Su (SU), Dimitris Dechouniotis (NTUA), Costas Papadakis (NTUA), Aris Dadoukis (CERTH), Donatos Stavropoulos (CERTH), Ana Juan Ferrer (ATOS), Roman Sosa Gonzalez (ATOS), Joaquin Iranzo Yuste (ATOS), Rowshan Jahan Sathi (TUB), Alex Willner (TUB), Lucas Nussbaum (Inria), David Margery (Inria), Cedric Crettaz (MI)
V3	15/03/2022	Final version	Brecht Vermeulen (imec)
V4	16/03/2022	Submitted version	Peter Van Daele (imec), Maria Chiara Campodonico (Martel)

DISCLAIMER

The information, documentation and figures available in this deliverable are written by the **Federation for FIRE Plus (Fed4FIRE+)**; project’s consortium under EC grant agreement **732638** and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

COPYRIGHT NOTICE

© 2017-2022 Fed4FIRE+ Consortium

ACKNOWLEDGMENT



Co-funded by the European Union



Co-funded by the Swiss Confederation

This deliverable has been written in the context of a Horizon 2020 European research project, which is co-funded by the European Commission and the Swiss State Secretariat for Education, Research and Innovation. The opinions expressed and arguments employed do not engage the supporting parties.

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:		R
Dissemination Level		
PU	Public, fully open, e.g. web	X
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to FED4FIRE+ project and Commission Services	

** R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

EXECUTIVE SUMMARY

Fed4FIRE+ testbeds are in constant change and Fed4FIRE+ partners are constantly adapting their testbeds to the latest requirements. Also, the federation as a whole needs constant upgrading and this deliverable is the 3rd in a series of 3 deliverables describing the developments on the testbeds carried out in the framework of the Fed4FIRE+ federation. While originally this was intended to be a systematic set of 3 cycles, based on series of Open Calls experiments providing feedback and requirements for adaptations, this turned out to evolve in a continuous way. This deliverable follows the lines of the requirements and specifications described in deliverable D3.05.

This deliverable provides an overview of the developments in WP3 during the period 2020-2021 of the Fed4FIRE+ project. All normal operations developments (adding testbeds, fix bugs, adding small features, etc) are part of Work package 2 while Work package 3 is focusing on adding larger new functionalities to the federation and its testbeds.

WP3 consists out of the following tasks, which are also the sequence of sections in this deliverable:

- Task 3.1 is focusing on SLA and reputation for testbed usage
- Task 3.2 is focusing on Experiment-as-a-Service (EaaS), data retention and reproducibility of experiments
- Task 3.3 is targeting Federation monitoring and interconnectivity
- Task 3.4 works on Service orchestration and brokering
- Task 3.5 researches ontologies for the federation of testbeds

TABLE OF CONTENTS

- DOCUMENT REVISION HISTORY 2**
- DISCLAIMER 3**
- COPYRIGHT NOTICE 3**
- ACKNOWLEDGMENT 3**
- EXECUTIVE SUMMARY 4**
- TABLE OF CONTENTS 5**
- LIST OF FIGURES 7**
- LIST OF TABLES 8**
- ABBREVIATIONS 9**
- 1 INTRODUCTION 10**
- 2 SLA AND REPUTATION SERVICE 11**
 - 2.1 Testbeds Integration with reputation and Sla service 12
 - 2.1.1 Reputation Functional Requirements: 13
 - 2.2 API Documentation 13
 - 2.2.1 i2cat OFELIA Testbed Integration 13
 - 2.2.2 iris Testbed Integration 16
 - 2.2.3 Generic API Documentation 18
- 3 MAINTENANCE ON THE USER’S CERTIFICATE HANDLING FOR NEW PORTAL AUTHORITY 19**
 - 3.1 I2CAT OFELIA certificate handling 20
- 4 EAAS, DATA RETENTION AND REPRODUCIBILITY OF EXPERIMENTS 22**
 - 4.1 ESPEC and jFED CLI 2 22
 - 4.2 Experiment orchestration (expo) 26
 - 4.3 Jupyter notebooks for reproducible experiments 28
 - 4.3.1 Jupyter notebooks at imec’s GPU Lab testbed 28
 - 4.3.2 Jupyter notebooks on Inria’s GRID’5000 28
 - 4.4 Grid’5000 metadata bundler 38
 - 4.5 Distrinet (delay based fidelity monitoring of network emulation) 40
 - 4.5.1 Prerequisites 40
 - 4.5.2 Installation 40
 - 4.5.3 Usage 41
 - 4.6 Implementing SFA Support on an Established HPC-flavored Testbed GRID’5000: Lessons Learned 42
 - 4.6.1 Introduction 42
 - 4.6.2 Background on Grid’5000 43
 - 4.6.3 Implementing the Slice-based Federation Architecture in g5k 43
 - 4.6.4 Divergence between GENI and g5k 46
 - 4.6.5 Related Work 50

D3.6: Developments for the 3rd cycle



4.6.6 Conclusions 50

5 CENTRAL BROKER 51

6 RESOURCE RECOMMENDATION SERVICE (RRS) 54

7 AUTOMATED OPENSTACK DEPLOYMENT 55

8 CONCLUSIONS 57

LIST OF FIGURES

Figure 1: Starting an experiment in jFed with the option to ask for an SLA for this experiment 11

Figure 2: Zabbix Service and selected algorithm to support Fed4FIRE+ Virtual Machine Monitoring .. 16

Figure 3: Privacy Notice site for i2CAT OFELIA, with the user’s URN and consent details 21

Figure 4: ESPEC example with new functionality of 'direct' for literal blocks of data..... 22

Figure 5: ESPEC example with ansible scripting, supporting the new item 'galaxy' for ansible galaxy (pre-packaged units of work)..... 23

Figure 6: ESPEC for deploying Openstack 24

Figure 7: yaml action file for jFed CLI 2 25

Figure 8: Wizard function of jFed CLI 2 to help build yaml action files..... 25

Figure 9: Example of an ExpO orchestration definition (including the new 'environment') 26

Figure 10: Documentation of the ExpO orchestration definition file 27

Figure 11: Jupyter notebook at imec's GPULab testbed 28

Figure 12: Grid'5000 metadata bundler usage and example..... 38

Figure 13: Central Broker Architecture..... 51

Figure 14: Filtering based on resource technology and features..... 52

Figure 15: Portal view of resource selection (wireless mobile nodes with wifi a/b/g/n/ac) 52

Figure 16: Resource filtering based on testbed and hardware capabilities..... 53

Figure 17: Resource Recommendation Service Architectural Approach..... 54

Figure 18: Specific test for the Openstack ESPEC on the Federation Monitor website..... 55

Figure 19: Latest runs of the Openstack ESPEC..... 56



LIST OF TABLES

Table 1: SLA Reputation Functional Requirement –REPUTATION_01 13

Table 2: i2CAT availability API call for nodes 14

Table 3: i2CAT availability API call for a slice 15

Table 4: Zabbix service parameters IRIS testbed 16

Table 5: Sample JSON Post and Response from the IRIS testbed Zabbix API. 17

Table 6: GET reputation of specific testbed 18

Table 7: Get reputation for all testbeds 18

Table 8: Submit rating - update reputation/credibility 18

Table 9: Comparison between former (left) and current (right) end user certificates..... 19

Table 10: Grid'5000 API calls triggered by AMv3 API calls 45

ABBREVIATIONS

FIRE	Future Internet Research and Experimentation
JSON	JavaScript Object Notation
SLA	Service Level Agreement
SLO	Service Level Objective
XML	eXtensible Markup Language
WSAG	Web Service-Agreement
API	Application Programming Interface
XML-RPC	Extensible Markup Language Remote procedure call
REST	REpresentational State Transfer
AM	Aggregate Manager
QoS	Quality of Service
QoE	Quality of Experience
MVC	Model-View-Controller
O/RM	Object-relational mapping
GUI	Graphical User Interface
CLI	Command Line Interface
HRS	Hybrid Reputation System
KPI	Key Performance Indicator
FAHP	Fuzzy Analytic Hierarchical Process

1 INTRODUCTION

Fed4FIRE+ testbeds are in constant change and Fed4FIRE+ partners are regularly adapting their testbeds to the latest requirements. Moreover, the whole federation needs constant upgrading and this deliverable is the 3rd and final in a double series of 3 deliverables describing on one side the requirements and specifications for the testbeds (Deliverables D3.01, D3.03 and D3.05) and on the other side the developments carried out (Deliverables D3.02, D3.04 and this deliverable D3.06). While originally this was intended to be a systematic set of 3 cycles, based on series of Open Calls experiments providing feedback and requirements for adaptations, this turned out to evolve in a continuous way.

So this deliverable provides an overview of the developments in WP3 during the period 2020-2021 of the Fed4FIRE+ project based on the requirements and specifications described in deliverable D3.05.

All normal operations developments (adding testbeds, fix bugs, adding small features, etc) are part of Work package WP2 while Work package WP3 is focusing on adding larger new functionalities to the federation and its testbeds.

WP3 consists out of the following tasks, which are also the sequence of sections in this deliverable:

- ➔ Task 3.1 is focusing on SLA and reputation for testbed usage
- ➔ Task 3.2 is focusing on Experiment-as-a-Service (EaaS), data retention and reproducibility of experiments
- ➔ Task 3.3 is targeting Federation monitoring and interconnectivity
- ➔ Task 3.4 works on Service orchestration and brokering
- ➔ Task 3.5 researches ontologies for the federation of testbeds

The sections of this document are linked to these tasks in the following order:

- ➔ Section 2 “SLA Reputation Service” is linked to the Task 3.1
- ➔ Section 3 “Maintenance of the User’s Certificate Handling” is linked to Task 3.1
- ➔ Section 4 “EaaS, Data Retention and Reproducibility of Experiments is linked to Task 3.2
- ➔ Section 5 “Central Broker” is linked to Task 3.4
- ➔ Section 5 “Resource Recommendation Service” is linked to Task 3.5
- ➔ Section 6 “Automated Open Stack Deployment” is linked to Task 3.5

2 SLA AND REPUTATION SERVICE

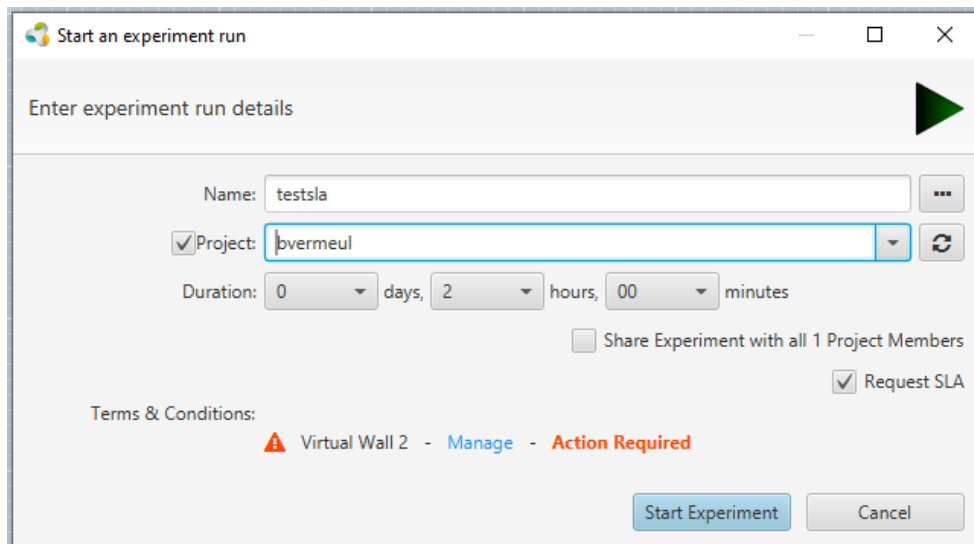
In the Fed4FIRE+ environment, the Service Level Agreement (SLA) and the Reputation Service provide the necessary tools and mechanisms for delivering to the users a quantitative view of the trustworthiness of the federated testbeds. This service facilitates the Fed4FIRE+ users to select the appropriate testbeds in the federation according to their experiment's requirements and the testbeds provide SLA on specific QoS metrics.

The aim of adding SLA within Fed4FIRE+ is to enable testbed providers to create offerings that experimenters can accept establishing an agreement with the testbed owner. We can understand the agreement as a contract between the platform providers and the testbed users. Once the agreement has been created, it must be verified that it is being fulfilled. The information related to the execution of an experiment, i.e., if there is an agreement violation, will be sent to the other components using a notification / subscription pattern.

The Reputation Service of Fed4FIRE+ aims to enhance and extend the already-developed reputation service of Fed4FIRE+ project. The updated service will leverage Quality of Service (QoS) metrics, such as Availability, Latency etc., Quality of Experience (QoE) metrics, e.g., Usability and Documentation Readability, and SLA data in order to compute the degree of confidence of both experimenters and testbed. At the end of an experiment, the users will be prompted to give their feedback for the reserved testbeds in order to update the reputation score of the testbed and the credibility score of the experimenter. This process mitigates the effect of abnormal or malicious evaluations and guarantees that the testbeds' reputation score is fairly computed.

During the first cycle, the SLA and reputation services were developed. At the second cycle, these services were updated and technical guidelines were created for the integration with the core testbeds of the federation (D3.4). The SLA and reputation services were integrated with NTUA and NITOS testbeds.

During the third cycle, OFELIA and IRIS testbeds were connected with these services and the code in jFed was updated to reflect the latest version of the SLA and reputation code (Figure 1). The new portal still must be updated with the new code. This is planned before the end of the project.



The screenshot shows a window titled "Start an experiment run" with a close button. Below the title bar is a header "Enter experiment run details" with a green play button icon. The form contains the following elements:

- Name: text input field with "testsla" and a menu icon.
- Project: dropdown menu with "lvermeul" and a refresh icon.
- Duration: 0 days, 2 hours, 00 minutes.
- Share Experiment with all 1 Project Members: unchecked checkbox.
- Request SLA: checked checkbox.
- Terms & Conditions: "Virtual Wall 2 - Manage - Action Required" with a warning icon.
- Start Experiment: blue button.
- Cancel: grey button.

Figure 1: Starting an experiment in jFed with the option to ask for an SLA for this experiment

2.1 TESTBEDS INTEGRATION WITH REPUTATION AND SLA SERVICE

The i2CAT OFELIA and IRIS testbeds were integrated with the Reputation and SLA service. The NETMODE and NITOS testbed were integrated in previous cycles. For the integration process, NTUA and ATOS partners documented and provided specific instructions for the integration. An experiment is uniquely defined by the slice URN, its starting and ending time and the involved testbeds. This information must be sent to the service's components in order to assess any SLA violations and update the reputation score of each involved testbed. This can be done in a unified way, in the new portal of the federation. Through this portal, SLA agreements will be conducted, and the experimenters will rate the conducted experiments. Then, the SLA and Reputation Service receive the information and requests for the agreements and the ratings respectively, the services will request the monitoring data from the testbeds involved in the experiment based on the unique slice URN.

For the computation of the reputation score of a testbed, no reputation service components installation on the testbed is required. The only requirement is that the testbeds expose a monitoring data API for the procedure mentioned above. Such monitoring data APIs were developed on both i2CAT OFELIA and IRIS testbeds, guided by NETMODE, and will be documented later on. After the completion of the experiment, the user is prompted to evaluate all the involved testbeds and provide the ranking for the QoS and QoE metrics. Then, the reputation service receives the user's evaluation as a JSON POST request. The JSON body section referring to the user's evaluation will have the following format.

```
"user_eval":[{"Usability":"very high", "Sup Satisfaction":"very high",  
"Doc Readability":"very high", "Operability":"very  
high","Availability":100, "Response":87},{  
"Usability":"very high", "Sup  
Satisfaction":"very high", "Doc Readability":"very high",  
"Operability":"very high","Availability":100, "Response":55}]
```

The above example contains two evaluations, since the experiment used resources from two testbeds and the experimenter must rate every testbed involved.

The Reputation Engine computes the new values of the testbed's reputation and the user's credibility using the evaluation received from the experimenter while the monitoring data are requested sequentially from the involved testbeds APIs. The updated testbed's reputation score will be available on the portal and combined with the definitions of the KPIs in order to facilitate future experimenters on the resource selection and compare different testbeds based on their performance on specific metrics.

The integration of a testbed with the SLA service requires initially the installation of the SLA management module. This component includes several sub-components, such as the Repository and the Assessment modules, that are responsible for maintaining the information about the agreements, the penalties, the violations and the templates and for assessing the QoS performance of the service based on specific KPIs. The SLA management module collects the monitoring data either periodically or aggregated at the end of the experiment in order to assess the provided service from the monitoring data with the exact same process described previously. The only difference is the interval of the monitoring data requests. These components were installed and configured on i2CAT-OFELIA and IRIS testbeds with the assistance of ATOS.

2.1.1 Reputation Functional Requirements:

ID	REPUTATION_01
Title	Reputation Service access to monitoring and SLA data
Short description	The new reputation computation engine access through APIs the monitoring data of an experiment and the information about SLA agreements and violations in order to calculate the user's credibility and readjust the user's evaluation if needed. More specifically, monitoring data and SLA data are compared with the experimenter's evaluation in order to adjust the credibility and the evaluation.
Additional information	The monitoring data are retrieved from the monitoring data REST API of each testbed while the SLA data are retrieved through the SLA Collector.
Type	DATA
Priority Level	High
Identified by Partner(s)	NTUA
Status	Completed

Table 1: SLA Reputation Functional Requirement –REPUTATION_01

2.2 API DOCUMENTATION

2.2.1 i2cat OFELIA Testbed Integration

In order to integrate with the reputation and SLA service, the i2CAT OFELIA testbed has developed a number of components. Specifically, a monitoring system and a set of internal and external interfaces help achieving this.

The development of the monitoring system allows the operator to configure the periodicity of a daemon job (by default this is set to 5 minutes). This system will first interact with a new internal testbed API, in charge of exposing the resources contained by a slice to other elements in the testbed. This will poll the list of available slices, obtain the resources via its IP (which also acts as their unique ID in the testbed) and proceed to query them in order to identify their availability through the execution of a simple ping. This is the metric of choice, since both the computing and the networking devices are reachable by their IP. On the other hand, the availability is determined in this manner since it is an operation internal to the testbed, operating in a private range of IPs, and thus not subject to external network segments or intermediate proxies or gateways that are out of the testbed control, in case any failure arose. Finally, this information is persisted in a database, keeping proper relation of the ID (IP) of the resource, the specific time of the availability check and its result, as well as its relation to the container slice.

A number of HTTP REST APIs interact with the monitoring system introduced above. First, the internal API allows the monitoring system to fetch which slices and resources are available, as previously registered in the testbed during the creation of an experiment. This runs in an internal system, co-located with the control stack of the testbed. Then, the external API makes it possible for the reputation and SLA system to query the availability data — previously obtained and registered in the database by the monitoring service. The latter API has two available endpoints that allow obtaining (i) the availability of a set of resources(s), given their URN or ID; and (ii) the availability of a set of resources, given the main slice URN. The external API is located at an external node, communicated with the internal testbed and exposing its endpoints from a public FQDN (f4f.lab.i2cat.net).

The tables below (Table 2 and Table 3) describe the two endpoints from this external API.

POST http://f4f.lab.i2cat.net:8449/rpc/availabilitymul

Description: Retrieve availability monitoring data of the requested resource(s)

Request example:

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" \
-X POST http://f4f.lab.i2cat.net:8449/rpc/availabilitymul -d '{
    "nodes": [
        "urn:publicid:IDN+ilabt.imec.be:i2cat-staff+node+10.216.12.27",
        "urn:publicid:IDN+ilabt.imec.be:i2cat-staff+node+10.216.12.28"
    ],
    "rstart": "2021-12-08 13:55:00 +0200",
    "rend": "2021-12-08 14:35:00 +0200"}'
```

Response example: (Content-type: application/json):

```
[
  {
    "urn:publicid:IDN+ilabt.imec.be:i2cat-staff+node+10.216.12.27": "1.000"
  },
  {
    "urn:publicid:IDN+ilabt.imec.be:i2cat-staff+node+10.216.12.28": "1.000"
  }
]
```

Response Representations: i) Success

Code 200 (OK):

ii) Error codes and messages

Code 400 (Bad request):

- "Content-Type must be set to "application/json""
- "Invalid requested period of time (start > end)"
- "Some argument is missing from: [rstart, rend]"
- "Arguments: the name from one of the following is needed: [nodes, node_id]"
- "Arguments: [nodes, node_id] are both provided and conflicting. Pick one"

Code 500 (Internal server error)

Table 2: i2CAT availability API call for nodes

POST http://f4f.lab.i2cat.net:8449/sla/monitor

Description: Retrieve availability monitoring data for all resources within the requested slice

Request example:

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" \
-X POST http://f4f.lab.i2cat.net:8449/sla/monitor -d '{
    "slice_urn": "urn:publicid:IDN+ilabt.imec.be:i2cat-staff+slice+ct1",
    "rstart": "2021-12-08 13:55:00 +0200",
    "rend": "2021-12-08 14:35:00 +0200"}'
```

Response example: (Content-type: application/json):

```
{
  "10.216.12.27": "1.000",
  "10.216.12.28": "1.000"
}
```

Response Representations: i) Success

Code 200 (OK):

ii) Error codes and messages

Code 400 (Bad request):

- "Content-Type must be set to "application/json""
- "Invalid requested period of time (start > end)"
- "Some argument is missing from: [rstart, rend]"
- "Some argument is missing from: [slice_urn, rstart, rend]"
- "No resources associated to slice_urn: <...>"
- "slice_urn=<...> is not valid"

Code 500 (Internal server error)

Table 3: i2CAT availability API call for a slice

2.2.2 Iris Testbed Integration

The IRIS testbed utilises the Zabbix framework to monitor Fed4FIRE+ virtual machine instances created by experimenters and provide experiment SLAs. Zabbix is an enterprise-class open-source network and application monitoring tool capable of monitoring millions of metrics. Zabbix automatically monitors the availability of discovered virtual machine instances on the Iris testbed network utilising the ICMP protocol. To support this activity, Zabbix tracks the reachability and unreachability of all the internal IRIS Testbed network IP addresses. When an experiment virtual machine (VM) become reachable, ICMP reply start and end time are logged, which is stored by Zabbix in a MySQL database.

To enable SLA data collection for experimenter VMs at the Iris testbed, we utilize the *Service Monitoring* capability in Zabbix. Service monitoring is intended for high-level view of monitored infrastructure and enables IRIS to provide the accessibility of a virtual machine instances for the duration of an experiment. Figure 2 shows how to configure the Zabbix service monitoring tool to support VM instances SLA tracking. At the lowest level of services supporting SLA monitoring are triggers. As shown in the Figure 2, the trigger at IRIS is the IP address “unavailable by ICMP ping”. The status of individual nodes is affected by the status of their triggers (i.e., reachable via ping, or not reachable via ping). Table 4 provides an overview of the service parameters utilised by IRIS in Zabbix.

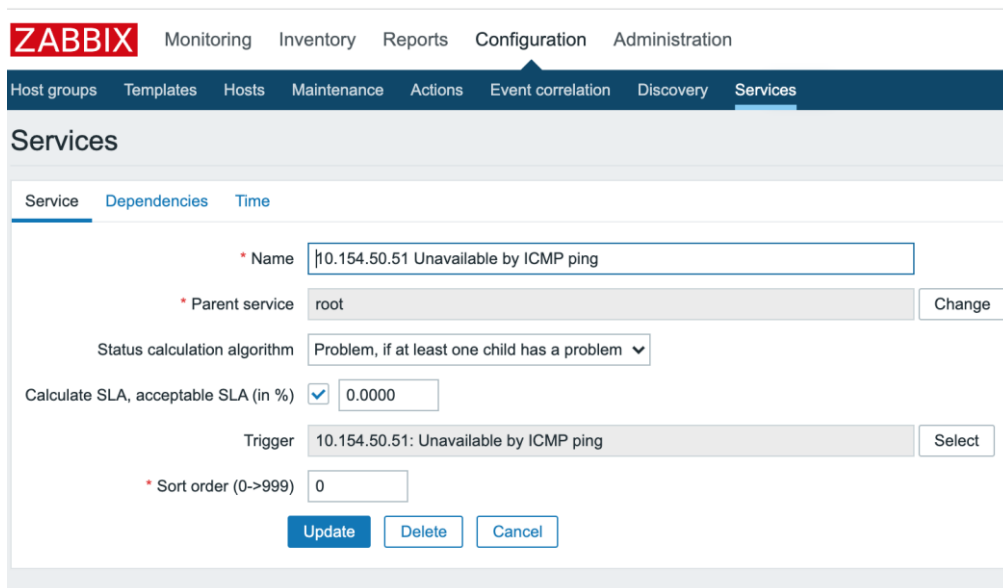


Figure 2: Zabbix Service and selected algorithm to support Fed4FIRE+ Virtual Machine Monitoring

Parameter	Description
<i>Name</i>	Service name.
<i>Parent service</i>	Parent service the service belongs to ('root' in our case)
<i>Status calculation algorithm</i>	Method of calculating service status
<i>Calculate SLA</i>	Enable SLA calculation and display.
<i>Acceptable SLA (in %)</i>	SLA percentage that is acceptable for this service. Used for reporting.
<i>Trigger</i>	trigger depends on the trigger status At IRIS the trigger is “Unavailable by ICMP ping”
<i>Sort order</i>	Sort order for display, lowest comes first.

Table 4: Zabbix service parameters IRIS testbed

2.2.3 Generic API Documentation

The reputation service REST API is developed to provide the essential information for every testbed. The Fed4FIRE+ portal can connect to this API to retrieve reputation scores of all testbeds and submit new evaluations after the completion of an experiment. The portal prompts the user to evaluate the Testbeds used in the experiment with an appropriate rating form through the GUI and submit this evaluation to the Reputation Service’s REST API along with information for the testbeds and resources used. The Reputation Service’s API returns to the portal the updated reputation values of the evaluated testbeds. In the following tables (Table 6, Table 7 and Table 8), three examples of the reputation service REST API calls are shown.

Description	Get reputation of specific testbed
Endpoint	/reputation/show
Type	GET
Example Body (url parameter)	/reputation/show?testbed=urn:publicid:IDN+ple+authority+cm

Table 6: GET reputation of specific testbed

Description	Get reputation of specific testbed
Endpoint	/reputation/showrep
Type	GET
Example Body	

Table 7: Get reputation for all testbeds

Description	Submit rating - update reputation/credibility
Endpoint	/reputation/userqoe
Type	POST
Example Body	<pre>{ "user_urn": " f4f_customer", "slice_urn": " slice_urn ", "start_time": "2018-11-05T13:25:00+02:00", "end_time": "2018-11-05T14:55:00+02:00", "resources": ["provider1", "provider2"], "user_eval": [{"Usability": "very high", "Sup Satisfaction": "very high", "Doc Readability": "very high", "Operability": "very high", "Availability": 100, "Response": 87}, {"Usability": "very high", "Sup Satisfaction": "very high", "Doc Readability": "very high", "Operability": "very high", "Availability": 100, "Response": 55}] }</pre>

Table 8: Submit rating - update reputation/credibility

3 MAINTENANCE ON THE USER'S CERTIFICATE HANDLING FOR NEW PORTAL AUTHORITY

During the third cycle, a number of Fed4FIRE+ testbeds introduced development efforts related to the maintenance on the user's certificate handling.

The Fed4FIRE+ Portal leverages a centralised Certificate Authority (CA). This CA changed its certificate in January 2020 (with the new authority/portal), which also introduced some changes on the structure of the X509 certificates issued to the end users of the Fed4FIRE+ federation. These changes are relevant to the fully federated Fed4FIRE+ testbeds, since they have to provide a Privacy Notice page to the end users with explicating indications on the processing of their data, requesting explicit consent and for a limited amount of time – upon which consent would be requested again. This page is loaded for the first time using a specific testbed, or upon expiry, to an end-user that leverages jFed for creation of experiments.

The following list details the changes incurred by the new certificates:

- ➔ Hardening of the RSA key length: from 1024 to 2048 bits.
- ➔ Change of issuer's fields: updating coordinators' details to the most up-to-date version. Specifically, on the "O", "OU", "GN" fields.
- ➔ Specifically on the introduction of "GN", "SN" and the update of "O" and "OU" fields.
- ➔ Changes in the "X509v3 extensions" fields: for instance, making explicit the expected usage in the "X509v3 Extended Key Usage" field and updating the "URI" field from the "X509v3 Subject Alternative Name" field.

Table 9 presents a side-to-side comparison between previous and current certificates, considering a dummy user of name "Jane" and surname "Doe" that works for a research or academic organisation of name "ResearchLab".

Former	Current
<p>Certificate:</p> <p>Data:</p> <p>Issuer: C = BE, ST = OV, L = Ghent, O = iMinds - ilab.t, OU = Certificate Authority, CN = boss.wall2.ilabt.iminds.be, emailAddress = vwall-ops@atlantis.ugent.be</p> <p>Subject: C = BE, ST = OV, O = iMinds - ilab.t, OU = iminds-wall2.janedoe, CN = <uuidA1>, emailAddress = jane@wall2.ilabt.iminds.be</p> <p>X509v3 extensions:</p> <p>X509v3 Subject Alternative Name:</p> <p>URI:urn:publicid:IDN+wall2.ilabt.iminds.be+user+jane, email:jane@wall2.ilabt.iminds.be, URI:urn:uuid:<uuidA2></p>	<p>Certificate:</p> <p>Data:</p> <p>Issuer: O = ilabt.imec.be, OU = authority, GN = ma, CN = <uuidB1>, emailAddress = None</p> <p>Subject: CN = <uuidB2>, emailAddress = jane.doe@researchlab.edu, GN = Jane, SN = Doe, O = ResearchLab, OU = ACADEMIC, dnQualifier = fed4fire</p> <p>X509v3 extensions:</p> <p>X509v3 Subject Alternative Name:</p> <p>email:jane.doe@researchlab.edu, URI:urn:publicid:IDN+ilabt.imec.be+user+jane, URI:urn:uuid:<uuidB2></p>

Table 9: Comparison between former (left) and current (right) end user certificates

3.1 I2CAT OFELIA CERTIFICATE HANDLING

Following the changes on the generation of the X509 certificates that have been introduced previously, two main efforts took place in the i2CAT OFELIA testbed.

The first one, related to the deployment of both trusted and exposed X509 certificates, required the following steps:

- ➔ updating the trusted CA certificate from Fed4FIRE+ to the one exposed by the Portal (available at https://portal.fed4fire.eu/root_certificate); and
- ➔ exposing the full certificate chain for the certificate exposed by the i2CAT privacy notice website (available at <https://f4f.lab.i2cat.net/privacy/>), instead of leaving the browser to auto-download the missing certificates.

The former is needed due to the expiration of the previous CA certificate, whereas the latter is expected by the built-in Java browser that is in use by the jFed client.

The second change has to do with the parsing of the end user X509 certificates. Specifically, two phases require parsing the fields from the certificates:

- ➔ the presentation of the Privacy Notice website to the end user and the registration of its consent (or lack of), persisting some ID provided by the end user's certificate; and
- ➔ assuming consent: the creation of the experiment and the release of the end user's on-the-fly generated credentials, whose identifier has to be matched to the aforementioned one.

The parsing method used until now, and deprecated after the new parsing mechanism, obtained first the environment's `SSL_CLIENT_S_DN` section (as received by the Nginx reverse proxy) and then extract the expected fields, following this format:

```
/C=.../ST=.../O=.../OU=.../CN=.../emailAddress=...
```

The "`emailAddress`" field was extracted in order to generate the user's URN, whose format is:

```
urn:publicid:IDN+<fed_name>+user+<user_name>
```

```
(e.g., urn:publicid:IDN+wall2.ilabt.imind.be+user+carolina)
```

This is so because the e-mail address is a local identifier for the Fed4FIRE+ federation (e.g., `carolina@wall2.ilabt.imind.be`). With this, the registration of the user's URN and its decision takes place. Assuming the end user consents to the processing of its data, the second phase starts: the experiment is created, and the on-the-fly generated GENI credentials

<https://github.com/GENI-NSF/geni-docs/blob/master/GeniApiCredentials.adoc>)

for the user are sent to the testbed. Now, the `<owner_urn>` field is available and can match against the URN generated beforehand.

The current method obtains the environment's `HTTP_SSL_CLIENT_S_DN` section, in order to retrieve similar data, and now parses its content according to the format

```
dnQualifier=..., OU=..., O=..., SN=..., GN=..., emailAddress=..., CN=...
```

It then creates the URN in a similar manner, even though the contents will differ. This is so because the e-mail address is no longer the identifier local to the Fed4FIRE+ federation, but the real e-mail of the end user. In this case, that would mean the URN is

```
urn:publicid:IDN+i2cat.net+user+carolina.fernandez
```

In the second phase, the GENI credential required further parsing, since the previous field was apparently not matching the generated URN. Instead, the e-mail address is obtained from this credential by first obtaining the `<owner_gid>` data, loading the embedded X509 certificate with specific libraries and retrieving the end-user e-mail address.

Figure 3 shows the generated URN and the Privacy Notice site, both visible to the end user. This can be accessible both from jFed and from any browser that has loaded the appropriate X509 certificate issued by the Fed4FIRE+ Portal.

i2CAT OFELIA testbed
Visiting user: `urn:publicid:IDN+i2cat.net+user+carolina.fernandez`

Testbed access: **Allowed**
(valid until 2022-06-21 15:42:03)

Terms & Conditions

Preliminary information
Updated: 20th February, 2019
Testbed: OFELIA (i2CAT)

Figure 3: Privacy Notice site for i2CAT OFELIA, with the user's URN and consent details

4 EAAS, DATA RETENTION AND REPRODUCIBILITY OF EXPERIMENTS

4.1 ESPEC AND JFED CLI 2

The Experiment Specification format is not a replacement for the RSpec format. An ESPEC contains an RSpec and combines it with other files.

- The purpose of an RSpec is to define which resources are needed.
- The purpose of an ESPEC is to additionally define which files should be placed where, and which scripts should be started.

The current ESPEC specification already allows some complex ESPECs. However, the base idea when using an ESPEC should be to keep it simple, and to put as little as possible in the ESPEC. It is meant for easily bootstrapping your experiment. It is not meant for running experiment logic.

It is preferable to keep ESPECs so simple that a user not familiar with the format, will be easily able to manually execute your experiment using tools that do not support ESPEC.

For a full description, see <https://jfed.ilabt.imec.be/espec>.

During this period, we extended the functionality, see some complex examples below:

```
version: 1.0-basic
rspec:
  - bundled: 3-nodes.rspec
upload:
  - exp-files-set1.tar.gz
  - bundled: exp-files-set2.tar.gz
    path: /tmp
    nodes: [central, exp1]
  - download: http://example.com/exp-files-set3.tar.gz
  - direct: |
      You can also directly specify the content of a file. This text will thus be stored on all nodes in /tmp/
      Check the yaml syntax of "literal-blocks" for details about syntax and removing indentation
    path: /tmp/demo.txt
execute:
  - bundled: setup-central-node.sh
    nodes: central
  - bundled: setup-exp-node.sh
    nodes: [exp1, exp2]
  - local: /work/repo/start-exp.sh
    nodes: [exp1, exp2]
```

Figure 4: ESPEC example with new functionality of 'direct' for literal blocks of data

```
version: 1.0-basic
rspec: my-experiment.rspec
dir:
  - path: /work/ansible/
    content: ansible
ansible:
  host:
    type: EXISTING
    name: control
    galaxy-command: /usr/local/bin/ansible-galaxy
    playbook-command: /usr/local/bin/ansible-playbook
    execute:
      - my-custom-ansible-install.sh
  galaxy:
    - download: http://example.com/ansible-requirements.yml
    - my-ansible-requirements.yml
  playbook:
    - bundled: setup-software.yml
      debug: 2
    - run-1st-experiment.yml
    - run-2nd-experiment.yml
  group:
    servers:
      - server1
      - server2
    clients:
      - client1
      - client2
```

Figure 5: ESPEC example with ansible scripting, supporting the new item 'galaxy' for ansible galaxy (pre-packaged units of work)

Most of the new features were needed to support the definition of an ESPEC for automatically deploying an Openstack deployment using EnOS.

D3.6: Developments for the 3rd cycle



```
version: 1.0-basic
rspec: enos3.rspec

upload:
- generated: keypair
- meta: experiment-info.json
- bundled: deployment
  nodes: [node0]

execute:
- bundled: enable-internet.sh
  sudo: true
- bundled: maximize-sda1.sh
  sudo: true
- bundled: add-sshkey-to-root.sh
  sudo: true
- bundled: force-python3-install.sh
  sudo: true
- direct: |
  #!/bin/bash
  echo "Remove references to 192.168.10.x subnet in /etc/hosts because rabbitmq fails to resolve hostname otherwise"
  sed -i.bak 's/192.168.10./192.168.0./' /etc/hosts
  sudo: true
- direct: |
  #!/bin/bash
  whoami

  cd deployment
  chmod +x install-enos.sh
  sudo ./install-enos.sh

  sudo patch /usr/bin/geni-get < geni-get.patch

  chmod +x setup-os-nat.py
  sudo ./setup-os-nat.py
nodes: [node0]

- direct: |
  #!/bin/bash
  source /opt/enos*/admin-openrc

  ~/.pyenv/versions/enos-venv/bin/openstack flavor list
  ~/.pyenv/versions/enos-venv/bin/openstack image list
  ~/.pyenv/versions/enos-venv/bin/openstack network list
  ~/.pyenv/versions/enos-venv/bin/openstack server create --flavor m1.tiny --image cirros.uec --network public test1

nodes: [node0]
```

Figure 6: ESpec for deploying Openstack

Together with the ESpec upgrades, we developed also jFed CLI 2 (command line version of jFed, <https://doc.ilabt.imec.be/jfed-documentation-5.9/otherjfedtools.html#experimenter-cli-2>) which now supports also ESpecs. In that way it is possible to fully automate very complex experiments.

This jFed CLI 2 now runs with yaml action files which make it very easy to describe more complex actions. The actions also resemble better the jFed GUI workflow while jFed CLI 1 worked closer to the underlying testbed API (AM API).

This is how a command is run:

```
user@laptop ~/Downloads/jfed_cli $ java -jar experimenter-cli2.jar --action run_experiment.yml -p login.pem
```

And this is a yaml file to describe an experiment of one docker container running for 120 minutes. Below you can find how the wizard function works to help build an action file.


```
action: RUN
experiment:
  requestRSpec:
    source: PROVIDE_CONTENT
    providedContentSource: |
      <rspec xmlns="http://www.geni.net/resources/rspec/3" type="request">
        <node client_id="node0" exclusive="false" component_manager_id="urn:publicid:IDN+docker.ilabt.imec.be+authority+am">
          <sliver_type name="docker-container"/>
        </node>
      </rspec>
  runLinkTest: true
  slice:
    sliceName: exp1
    failOnExistingSlice: false
    expireTimeMin: 120
    projectSource: PROVIDED
    project: myProject
  waitForReady:
    maxTimeMin: 5
  shareWith:
    projectMembers: true
  deleteOn:
    failCreate: true
    failBecomeReady: true
    failConnectivityTest: true
    failLinkTest: true
```

Figure 7: yaml action file for jFed CLI 2

The easiest way to create an action file, is to use the builtin "wizard". Start it like this:

```
user@laptop ~/Downloads/jfed_cli $ java -jar experimenter-cli2.jar --wizard
Welcome to the jFed CLI Wizard. This tool guide you through all the available actions and options.
This will only create the YaML config for the action, it will not execute the action.
At the end, it will be shown how the created action can be executed.

Available actions:

0) run
1) createSlice
2) performOperationalAction
...
```

Figure 8: Wizard function of jFed CLI 2 to help build yaml action files

4.2 EXPERIMENT ORCHESTRATION (EXPO)

ExpO is short for "Experiment Orchstrator". It allows you to run time-sensitive experiments over multiple machines.

ExpO consists out of two pieces of software:

- ➔ the **ExpO slave** which is present on all machines participating in the experiment, waiting for instructions on when to execute commands
- ➔ the **ExpO director** which executes experiments defined in an [Experiment Orchestration definition](#)

See for the full details at: <https://gitlab.ilabt.imec.be/ilabt/expo>

An example of an orchestration definition:

```
version: 1.0
nodes:
  node1: [groupA]
  node2: groupB
  node3:
    - groupA
    - groupB
  node4
commands:
  - command: "iperf -s"
    groups: [groupA]

  - after: 10
    id: iperf_client
    command: "iperf -c server"
    groups: [groupB]

  - command: echo "Hello $EXAMPLE_NAME"
    args:
      chdir: /tmp
      environment:
        EXAMPLE_NAME: 'World'
    nodes: node4
```

Figure 9: Example of an ExpO orchestration definition (including the new 'environment')

And all functionality is described here:

The file format

version Currently always 1.0

nodes List of nodes expected in the experiment. Optionally you can specify to which groups the node belongs in this experiment

commands : List of commands to be executed

Command format

command : the command to execute

after : number of seconds after the previous command that this command must be executed (optional, if omitted, it will be executed immediately after the previous command)

daemon : whether to keep this command running in the background. Used by the CLI to know if it should wait for the command to finish or not.

groups : the groups which must execute the command (optional if **nodes** has been specified)

nodes : the nodes which must execute the command (optional if **groups** has been specified)

id : a custom ID to identify MQTT-messages which relate to this command, such as the result, stdout, stderr, stdin (optional, if omitted, the index of this command in the **commands** -list is used as an id)

stderr : whether to stream the stderr as MQTT-messages with topic `expo/<experiment_id>/node/<node_id>/cmd/<command_id>/stderr` (defaults to True)

stdout : whether to stream the stdout as MQTT-messages `expo/<experiment_id>/node/<node_id>/cmd/<command_id>/stdout` (defaults to False)

stdin : whether to stream data sent to MQTT topic `expo/<experiment_id>/node/<node_id>/cmd/<command_id>/stdin` or `expo/<experiment_id>/group/<group_id>/cmd/<command_id>/stdin` to the stdin of the process (defaults to False)

Warning: the order in which data is streamed to the stdin of a process is indeterminate when mixing both topics

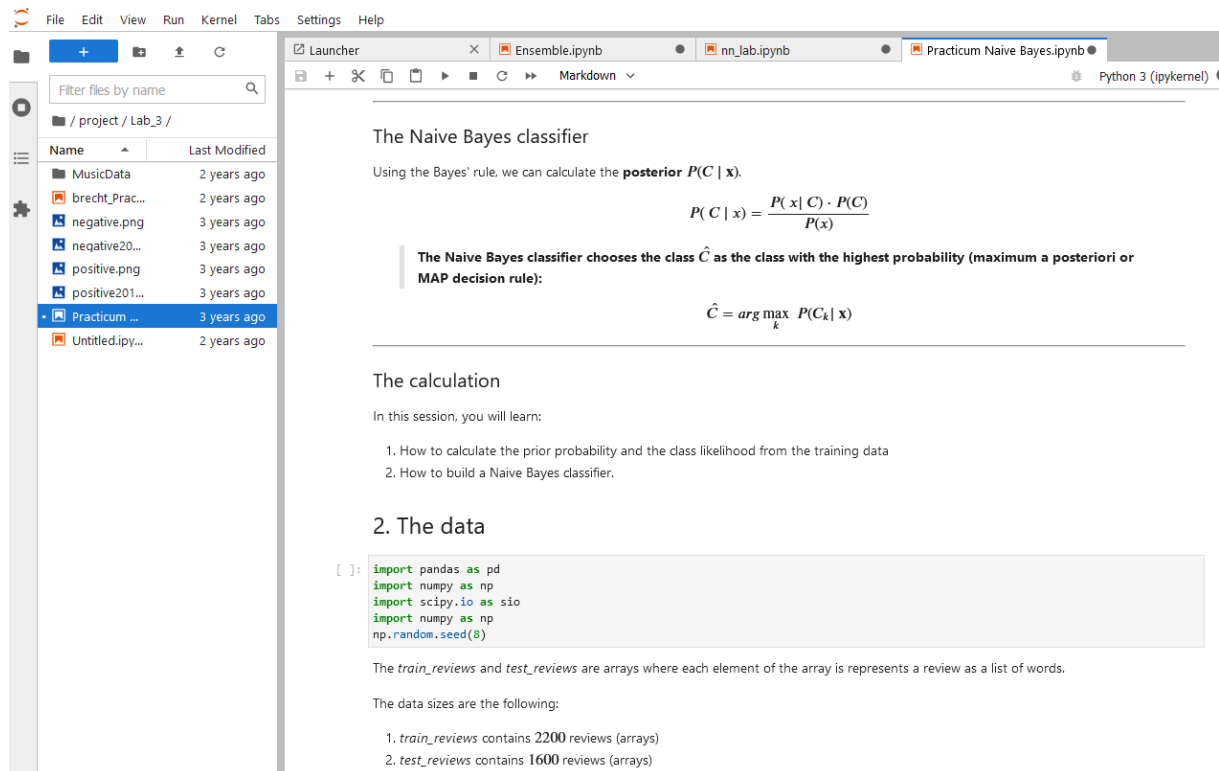
Figure 10: Documentation of the ExpO orchestration definition file

4.3 JUPYTER NOTEBOOKS FOR REPRODUCIBLE EXPERIMENTS

4.3.1 Jupyter notebooks at imec's GPULab testbed

As shown here <https://doc.ilabt.imec.be/ilabt/jupyter/index.html>, you can select GPUs or only CPUs to use the notebooks. This is typically used for first steps in machine learning, classes and quick experimentation (GPULab with a job-based workflow is then used for longer running more advanced jobs).

Below you can see a screenshot of the environment, with an example class on machine learning that has been used.



The Naive Bayes classifier

Using the Bayes' rule, we can calculate the **posterior** $P(C | \mathbf{x})$.

$$P(C | \mathbf{x}) = \frac{P(\mathbf{x} | C) \cdot P(C)}{P(\mathbf{x})}$$

The Naive Bayes classifier chooses the class \hat{C} as the class with the highest probability (maximum a posteriori or MAP decision rule):

$$\hat{C} = \arg \max_k P(C_k | \mathbf{x})$$

The calculation

In this session, you will learn:

1. How to calculate the prior probability and the class likelihood from the training data
2. How to build a Naive Bayes classifier.

2. The data

```
[ ]: import pandas as pd
import numpy as np
import scipy.io as sio
import numpy as np
np.random.seed(8)
```

The `train_reviews` and `test_reviews` are arrays where each element of the array is represents a review as a list of words.

The data sizes are the following:

1. `train_reviews` contains 2200 reviews (arrays)
2. `test_reviews` contains 1600 reviews (arrays)

Figure 11: Jupyter notebook at imec's GPULab testbed

4.3.2 Jupyter notebooks on Inria's GRID'5000¹

Computer science testbeds often require extensive experiment automation to be used efficiently. Jupyter notebooks can contain the scientific reasoning, experiment orchestration, and experiment results in an easy to read, easy to execute format. However, some level of support is required from the testbeds to facilitate notebook use on their platforms. Additionally, this format can be used for more than driving experiments, and different uses have different requirements in terms of support.

In this section we present an analysis of different notebook uses to be expected on computer science testbeds, as found through the feedback of users of the g5k testbed and a survey of other similar testbeds. Then we present the technical implementation of Jupyter on the g5k to support those uses.

¹ This can be found at <https://hal.archives-ouvertes.fr/hal-03233095> as well.

4.3.2.1 Introduction

Experiments in the study of distributed systems often require complex instrumentation to automate experiments involving large numbers of machines. On g5k, a computer science testbed based in France, we invite researchers to automate their experiments as much as possible. g5k's usage policy limits use of testbed resources during the workday, in order to ensure availability for small scale interactive experiments and preparatory work, and thus forcing large-scale experiments to be run at nighttime and during weekends.

Tools such as Execo² and EnOSlib³, are available to help users automate their experiments. However, automating experiments using a specialized tool is a heavy commitment, and users for the most part appear to automate their experiments using ad hoc scripts. Such scripts, built slowly through trial and error, are often written with no thought towards dissemination or reuse. As such they remain unpublished, and the experiment running process is often only vaguely described in the corresponding article.

Computational notebooks are systems inspired by literate programming, laboratory notebooks, and REPL (Read-Eval-Print-Loop) interactive shells, that mix in a single file prose detailing a scientific process, code implementing that process, and the output of that code. Notebooks are often mentioned as a possible tool against the current reproducibility crisis, for their ability to structure messy scripts into self-contained self-explanatory files. Because notebooks are code-agnostic, the code used in them would often be the same as the one used in ad hoc scripts. This means that the barrier of entry is significantly lower than for other forms of tooling.

Using a notebook does not make experiments reproducible in and of itself, and studies⁴ have shown that users must be careful in how they structure and use their notebook for it to be reproducible by other users. However, properly used, they are one of the simplest tools to increase the potential for reproducibility of their experiments. For these reasons we started looking at the possibility of adding Jupyter⁵, the most popular notebook tool suite, to the g5k testbed.

In order to decide what kind of support to add to our testbed we surveyed how other computer science testbeds implemented their own Jupyter support. We also requested feedback from g5k users concerning their use, real or intended, of Jupyter notebooks on the testbed. The analysis of user feedback and the support for Jupyter in other testbeds is what guided our implementation of Jupyter in order to facilitate as many uses as possible.

In this section we will present the different uses of notebooks we derived from our surveys, their needs and constraints, and discuss how they apply to testbeds in general and g5k specifically. We will then explain which uses we aim to support and how we implemented a JupyterHub installation that satisfies those uses, as well as share the technical difficulties we encountered as a result of Jupyter's architecture.

The section is structured as follows. Section 0 presents g5k and the Jupyter stack we aim to implement. Section 4.3.2.3 details the notebook uses derived from user feedback, and how they apply to testbeds. Section 0 discusses the integration of Jupyter in other testbeds. Finally, we discuss technical details of our implementation and the technical difficulties encountered in Section 0 before concluding.

² M. Imbert et al., "Using the execo toolkit to perform automatic and reproducible cloud experiments," in CloudCom, 2013.

³ R. Cherrueau et al., "Enosstack: A lamp-like stack for the experimenter," in CNERT, 2018.

⁴ J. F. Pimentel et al., "A large-scale study about quality and reproducibility of jupyter notebooks," in MSR, 2019.

⁵ T. Kluyver et al., "Jupyter notebooks-a publishing format for reproducible computational workflows," in ELPUB, 2016.

4.3.2.2 Background

4.3.2.2.1 g5k

The g5k project⁶ was initiated in 2003 by the French HPC (High Performance Computing) research community to provide a large scale highly reconfigurable testbed in order to experiment at scale. Today g5k has evolved in a testbed similar to Chameleon⁷ or CloudLab⁸, covering topics like HPC, AI, Clouds, Edge Computing, Big Data. The services it offers are similar to those offered by the aforementioned testbeds⁹: bare metal provisioning, network isolation, monitoring services, etc. It also offers similar hardware: x86 and ARM servers, HPC networks, GPUs, etc.

g5k infrastructure: The g5k testbed is distributed over eight sites all over France and in Luxembourg and is maintained by a single technical team. Although user management, access control, and maintenance are performed globally, each site operates with their own clusters (sets of closely interlinked homogeneous nodes), site front-end, and service machines. Sites are interlinked by a dedicated network.

Users for the most part connect via `SSH` to one of g5k access points, from there they are able to connect to the site front-end. On the site front-end machine users have access to their site-specific homedir and the tools to reserve resources and deploy environments to these resources. g5k's testbed control infrastructure is based on OAR¹⁰ and associated tools. Using OAR, the users can book resources, going from single core to entire clusters, based on relative topologies, called *resource trees*, and filter on the resources' properties.

Alternatively, users can interact with the testbed through a REST API. Using the API users can list resources available on every site and interact with the instances of OAR of the different sites. User authentication on the API uses `http-basic-auth` with the user's g5k credential, except for calls made from one of the site front-ends which are automatically associated to the user using `identd`. This API is often used by tools that automate experiments.

4.3.2.2.2 Jupyter

During research a lot of code is often generated to collect and process data. Often this code was unpublished, researchers simply described the process in the corresponding article, and even when it was published the code was often relatively undocumented and hard to follow. In proposing their notebook format integrating code and prose, the Jupyter team hoped to foster readable code for reproducibility.

Jupyter notebooks combine ideas of literate programming, interactive programming, and laboratory notebooks. Notebook files contain code cells interspersed with text. Additionally, outputs generated by the executed code cells are also saved in the notebook. The jupyter-notebook application is designed to view, edit, and run notebooks and the execution of code cells is handled by Jupyter kernels.

Kernels are language-specific programs similar to interactive shells that execute the code contained in cells and keep program state (variables, functions, ...) in between cells. Because state is only kept in the kernel and not the notebook files, reproducibility of notebook outputs is only guaranteed if cells are executed exactly once and in order.

However, if users are careful about properly executing their notebook, and document their process in text cells, notebooks are good supports for reproducible research.

⁶ D. Balouek et al., "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, 2013

⁷ K. Keahey et al., "Chameleon: A scalable production testbed for computer science research," in *Contemporary High Performance Computing*, CRC Press, 2019.

⁸ R. Ricci et al., "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *login*, vol. 39, no. 6, 2014

⁹ L. Nussbaum, "Testbeds Support for Reproducible Research," in *ACM SIGCOMM 2017 Reproducibility Workshop*, 2017.

¹⁰ N. Capit et al., "A batch scheduler with high level components," in *CCGrid*, 2005.

Jupyter tool stack: The Jupyter notebook file format is a json file describing the different cells, their type (code, text, output), and their content, as well as some meta-data such as the kernel language to use, and the cells execution order for the saved outputs. In this article the term *notebooks* will refer to files in this format.

jupyter-notebook and **jupyter-lab** are applications used to provide a graphical user interface for notebooks. These application interfaces are accessed through a web browser and provide a file browser, an interface to view, edit, and execute notebooks, and ways to manage currently running kernels. **jupyter-notebook** is the first version of such application and **jupyter-lab** is a more recent successor. In this article we refer to these applications in their different forms as *labs*, so as to avoid confusion with notebooks.

JupyterHub¹¹ is a web application designed to facilitate the use of notebooks in multi-user environments, hereafter called *hub*. The hub handles user authentication, the starting and stopping of multiple lab instances at the user's request and transparently redirects the user to their lab interface. JupyterHub is structured around modular components that can be adapted to the hub operator's infrastructure.

The authenticator module controls user management and sets the authentication modalities. The default authenticator module uses Unix accounts on the machine, but other modules are available to using ldap or external OAuth providers.

The spawner module controls the lifecycle of lab instances. These lab instances, **jupyter-labhub**, are extensions of the standard lab applications that can only be executed conjointly with a hub. Different spawner modules can instantiate labs in different manners, from starting a new lab on the hub machine, to instantiating new labs in containers on a kubernetes¹² cluster, to using HPC task managers.

Additionally, the hub interacts with a reconfigurable http/websocket proxy that is capable of adding and deleting redirections. All incoming connections to the hub or labs are routed through the proxy with routes being added and removed by the hub as needed. For infrastructures where the lab and hub machines might not be routed to the internet, the proxy is the only component users need to be able to connect to.

4.3.2.3 User feedback, notebook uses, and objectives.

Our main objective when we started considering adding Jupyter to g5k was to offer an alternative for experiment scripts that would foster reproducibility. But before deciding the perimeter of what to implement we wanted to measure user interest and needs. We asked users for their use (existing or intended) of Jupyter Notebooks on g5k. Based on this feedback we established five different possible uses of notebooks on testbeds.

1. Notebooks as experiment drivers. These notebooks run the experiments from beginning to end, starting with resource reservations and going at least to data collection. To support this usage testbeds must provide an environment where resource reservation and interaction with reserved nodes are possible. Additionally, some form of storage will likely be necessary to store experiment bulk results, e.g. execution logs and relevant output. To maximize the cross-user reproducibility of such notebooks some attention must be given to the seamless authentication of users. Ideally resource reservations should not require user credentials to be written in the notebook. This can be achieved by preloading credential information in the environment the lab is executed in, or by providing an out-of-band way for the testbed to provide the resource management system with user information. Although this is the use case that had us interested in notebooks in the first place only one user reported interest in this approach during user feedback.

On g5k, site front-ends are the designated environment for such notebooks: they have access to the user site homedir, have connectivity to every node on the platform, and contain the necessary tools to perform resource reservations. Moreover, execution on the site front-ends allows for credential-less resource reservation since OAR commands derive user info from the callers Unix account and API calls are automatically matched to users using **identd**. However, using site front-ends implies providing users with the ability to choose which site they want to operate on since every site has a separate homedir.

¹¹ Project Jupyter team. (2016). Jupyterhub 1.3.0 documentation, [Online]. Available: <https://jupyterhub.readthedocs.io/en/stable/>.

¹² A. Verma et al., "Large-scale cluster management at Google with Borg," in EuroSys, 2015.

2. Notebooks as experimental payload. The code contained within these notebooks is the core of experiments. These notebooks run on the reserved resources, and either contain or control the computation that is the subject matter of the experiment. They run from the moment they started on the reserved resource until they output the relevant experiment results. To support this usage testbeds must provide a way to execute the notebooks on reserved resources. Testbeds with heterogeneous nodes should provide ways for the users to select on which nodes the notebooks are going to be executed. When virtualization or containerization is used, testbeds need to consider whether they want to provide ways for the user to dimension their resources. Additionally, some notebooks might require specific libraries, so testbeds should consider whether they need to provide multiple different environments or ways for users to add to the existing environments. During user feedback, two users indicated using notebooks in this fashion on g5k. We were also made aware that another user had shared a script meant to automate the setup for this usage.

On g5k, satisfying this usage implies providing a way for users to start lab instances inside arbitrary OAR jobs. To guarantee access to any resource the user might need, our installation will have to let the user select any site and resource they want. For environment adaptability we plan on adding Jupyter commands to environments where the user's site homedir is available, letting users customize the environment by adding libraries in their homedir.

3. Notebooks for post-processing. These notebooks are executed after an experiment to process the results. Supporting this usage will be dependent on your testbed's infrastructure and the type of post-processing expected. Some post-processing can be rather computationally intensive or require specific hardware and notebooks implementing these post-processing should be considered the same as experiment notebooks. Others can be executed on more standard hardware. In either case attention should be given to how users store their results, since these notebooks access experimental data. There was no direct testimony of this usage in user feedback.

On g5k, supporting this use case requires being able to execute notebooks on both front-ends and nodes. This is owed to the fact that some forms of storage are only accessible on specific nodes, and to the fact that heavy computations are not authorized on the site front-ends. As such this use case covers the same constraint as the two previous one for us.

4. Notebooks for exploratory programming. Notebooks for exploratory programming are used by users as a form of enhanced interactive shell in order create new code through trial and error. Most of the previously presented usages often start that way and are refined into fully functional notebooks over time. This use case is somewhat transversal to the previous ones as users preparing to run an experiment on a platform need to be able to test their notebooks interactively to smooth out all the kinks. Supporting this kind of usage is done through providing ways to access a lab interface that lets users benefit from the interactive component of notebooks. Although this usage is not always differentiable from the previous ones, two users reported using notebooks specifically to figure out parameters to specific machine learning algorithms before running the model in a non-notebook fashion.

On g5k, it is already possible to use a VPN or ssh tunneling to connect to lab applications running on front-ends or nodes. However, some users have expressed difficulties with using those tools, pushing us to explore the possibility of providing native web access for labs.

5. Notebooks as tutorials. Notebooks as tutorials are notebooks provided to the users by teachers that aim to present and explain to the users some specific concept. As with the previous usage these notebooks rely extensively on the interactive component of lab applications. Since students might not be familiar with the ways one interacts with a given testbed, this usage is best served by providing full web access to lab instances through a **JupyterHub**-like system. Teachers that use g5k have been using notebooks in their courses, and have expressed interest for a **JupyterHub** type access that would reduce the need to guide students through the node-reservation process, setting up the VPN, and manually installing and starting a Jupyter lab.

After considering the demands for these different use cases, we decided that g5k should implement a fully web-based access allowing users to start a lab on any arbitrary node or front-end from a single interface.

4.3.2.4 Related work

While planning our own implementation of Jupyter we searched for other computer science testbeds that implement Jupyter notebooks as a feature.

Chameleon, a testbed similar to g5k based on Openstack¹³, implemented their own Jupyterhub instance in 2019. In Chameleon's implementation JupyterHub¹⁴ starts labs in docker containers. Notebooks executed on these labs can perform resource reservations and drive experiments; Chameleon's implementation uses environment variables containing the user's Openstack credentials to facilitate this process. Chameleon also provides a notebook library for users to share experiments in.

The Minnesota Supercomputing Institute (MSI) also implemented a JupyterHub¹⁵ instance that runs lab instances on HPC computing nodes using the Torque batch scheduler.

Although they have not published any article on the subject, Jupyter is also available on the following testbeds:

- As part of the Fed4FIRE+ European testbed federation, imec's GPU Lab testbed proposes a docker powered JupyterHub instance¹⁶. Lab instances are executed in gpu enabled containers, and the hub offers different container images with a variety of software stacks (R, tensorflow, spark, julia). Users can request specific container sizes from the spawner page, and a separate interface can be used to active port-forwarding between the containers and the internet.
- GriCAD, the University of Grenoble computing center, has a JupyterHub¹⁷ instance capable of starting lab instances on a dedicated machine (website:). This testbed also supports running labs on computation nodes, but such instances are only accessible through ssh tunnelling.
- The IDRIS, a national supercomputing center in France with stringent security measures, bypasses the use of JupyterHub¹⁸ by providing a modified version of `jupyter-lab` that adds itself to a reconfigurable proxy at startup providing a way for users to access their lab instance (website:). These lab instances can be executed both on a front-end machine and computing nodes, but require manual reservation of resources using slurm beforehand.

To the best of our knowledge, g5k is unique in that we are implementing in a single instance of JupyterHub the ability to spawn lab instances both on reserved resources and front-ends machines, when other testbeds will usually do one or the other.

¹³ O. Sefraoui et al., "Openstack: Toward an open-source solution for cloud computing," International Journal of Computer Applications, 2012

¹⁴ J. Anderson and K. Keahey, "A case for integrating experimental containers with notebooks," in CloudCom, 2019.

¹⁵ M. Milligan, "Interactive HPC gateways with jupyter and jupyterhub," in PEARC, 2017.

¹⁶ imec iLab.t team. (2020). Jupyterhub at imec ilab.t - imec ilab.t documentation, [Online]. Available: <https://doc.ilabt.imec.be/ilabt/jupyter/>.

¹⁷ GriCAD. (2020). Les notebooks jupyter : User documentation for gricad services. French, [Online]. Available: <https://gricad-doc.univ-grenoble-alpes.fr/en/notebook/>

¹⁸ IDRIS. (2020). Idris - jean zay: Access to jupyter notebook and jupyterlab with tensorboard, [Online]. Available: <http://www.idris.fr/eng/jean-zay/pre-post/jean-zay-jupyter-notebook-eng.htm>

4.3.2.5 Implementing JupyterHub on g5k

4.3.2.5.1 General setup

We decided to deploy a single JupyterHub instance for the whole of g5k. This instance is placed on a service machine in the internal g5k network, with a reconfigurable proxy executed on the same machine as the hub. To allow users access to the reconfigurable proxy, and thus the hub and labs, a route is added to g5k's pre-existing Apache proxies. This frontline proxy is already used for other g5k services, can handle user authentication, and will mitigate any weakness the reconfigurable proxy might have.

The authentication module used in our hub is `jhub_remote_user_authenticator`¹⁹, which removes the authentication page and instead relies on incoming HTTP headers for authentication. This allows our frontline proxy to perform authentication and pass on authentication information to the hub through http headers. This is advantageous as it uses g5k already established authentication infrastructure without involving a new service.

The spawner module is a custom module implemented specifically to match g5k's needs, called `G5kSpawner`. From the user point of view the spawner requires the user select a site and whether to start the lab on a front-end or a node. If a node is selected, the users can specify the OAR resource tree to request, the walltime of the OAR job and other information required by OAR to service the request.

Spawner modules are required to implement three functions:

start used to start a new instance of the lab application, return the hostname (or ip) and port of the lab to the hub;

poll used to query the status of a given lab instance, returns nothing if the lab is still running or the lab's exit code if it is not;

stop used to shutdown an instance of the lab application, returns nothing.

Additionally, the spawner can store the information necessary to track lab instances across hub reboots to persistent storage.

`G5kSpawner` contains two different implementations of each of the three main functions and calls upon one or the other depending on whether the users request a node or a front-end.

4.3.2.5.2 Execution on nodes

For the execution of lab instances on compute nodes the hub delegates the execution of the `jupyter-labhub` program to OAR, that it controls through g5k's REST API.

The REST API allows the hub to interact with OAR instances of every site and as a service operated by g5k, the hub is provided an SSL client certificate allowing it to make calls to the API in the name of the user requesting the lab instance. When requesting resources from OAR the hub provides the lab command to execute. OAR will execute the command automatically once the requested resources become available on the main node of the request. If the `jupyter-labhub` command ends before the end of the OAR job, the job will be ended immediately by OAR. As such the lifecycle of the OAR job and the lifecycle of the lab instance are closely interlinked and the hub treats them as one and the same. Under this mode of operation the three main functions operate as follows:

start: The spawner selects a port on which to run the lab and prepares the lab command. This command along with OAR information such as the requested resource tree and walltime, are POSTed to the `/sites/<SITE>/jobs` endpoint to create a new OAR job, and the spawner gets a job-id from the reply. The execution site and job-id are saved to the spawner state and added to persistent storage. The spawner monitors the status of the job using the `/sites/<SITE>/jobs/<JOB-ID>` endpoint. Once the job is scheduled this endpoint also provides the hostname of the reserved node to the spawner which is returned along with the selected port to the hub.

¹⁹ C. Waldbieser et al. (2019). Jupyterhub remote user authenticator, [Online]. Available: <https://github.com/cwaldbieser/jhub-remote-user-authenticator>.

poll: The spawner checks the status of the OAR job using the `/sites/<SITE>/jobs/<JOB-ID>` endpoint. If the job state is *finished* or *error* the function returns 254 in lieu of an exit code. For any other status value, the lab is considered to be still running and the function returns nothing.

stop: The spawner issues a **DELETE** call to the `/sites/<SITE>/jobs/<JOB-ID>` endpoint, requesting the end of the OAR job.

The close interlink between the lab and the corresponding OAR job, the REST API, and the existence for the python-grid5000 library²⁰ used to interact with the API greatly simplifies the code used when instantiating labs on nodes.

4.3.2.5.3 Execution on site front-ends

For the execution of lab instances on a site front-end the hubs need to connect via ssh to the requested front-end and start the new lab process as the requesting user.

Unlike with the **REST API**, g5k has no preexisting method of acting on a front-end as an arbitrary user. And since the hub cannot be authorized to connect via ssh as any user, a new dedicated user was added to the front-end machines for the hub to connect to. This hub user is authorised to run a single script, `jupyterctl`, as root. The `jupyterctl` script act as a wrapper around Jupyter commands, performing the necessary user switching while limiting the capabilities of the hub user to the strict necessary. `jupyterctl` uses `sudo` to run `jupyter-labhub` and `kill` as other users while switching context to their homedirs. Additionally, the script will only agree to poll or stop Jupyter processes. In this mode the three main functions operate as follows:

start: The spawner prepares the environment variables and the arguments for the `jupyter-labhub` as usual but uses `sudo jupyterctl start` as a command instead. Additionally, the `port=` argument is removed by the spawner. The command is executed via ssh on the selected site front-end. On the front-end the `jupyterctl` command scans the machine for a free port and adds the corresponding `port=` argument. Then the `jupyterctl` script extracts the target user from the environment variables set by the hub and starts `jupyter-labhub` as the target user in their homedir, making sure to pass along all the environment variables and arguments set by the hub. Finally, the `jupyterctl` script outputs the `jupyter-labhub` process-id and selected port. The process-id and the selected site are added to the spawner's persistent state. The spawner returns the front-end hostname and the selected port to the hub.

poll: The spawner connects to the selected site and runs the `sudo jupyterctl poll <process-id>` command. The script outputs the number of Jupyter processes matching the process-id. If none were found the poll function assumes the lab instance is dead and returns 254 in lieu of exit code, otherwise the lab is still running, and poll return nothing.

stop: The spawner connects to the selected site and runs the `sudo jupyterctl stop <process-id>` command while passing the target user in an environment variable. The script checks that the target process-id is a Jupyter process and uses `sudo kill` as the target user. The script returns the number of matching processes after the kill, and the stop function returns nothing.

This code used in instantiating labs on front-ends is more complex and split between the G5kSpawner and the `jupyterctl` script. Part of these complexities is due to assumptions made by JupyterHub and a dead-lock between the hub and the lab. We will expand on these circumstances in a later section.

²⁰ M. Simonin. (2019). Python-grid5000, [Online]. Available: <https://gitlab.inria.fr/msimonin/python-grid5000>.

4.3.2.5.4 User experience

From the user point of view accessing the hub is done in the same way as any g5k interface. The same dialogue is used for authentication as in other services, and users do not need to log in a second time in **JupyterHub**. The single **JupyterHub** instance allows the user to interact with every site, and for node deployment the user is free to choose their resources using the same syntax as they would when connected via ssh. After pressing the start button, the user waits from a few seconds to a few minutes before being seamlessly redirected to the **Jupyter lab interface**.

The lab instance provided through **JupyterHub** comes with two kernels, one for python3 code and one for bash code. Users in need of different kernels can use `pip` to install them in their `homedir`. Kernels installed in this manner, as well as any python library the user may need, will become available in the lab environment.

Since labs on the front-ends are instantiated as the correct user, OAR commands and calls to the g5k API will work seamlessly without requiring any credential information. Because of this, no sensitive information needs to be stored in the notebook and notebooks should work without modification when shared between users.

Labs instantiated on nodes are started on the g5k standard environment, in which the user's site `homedir` is automatically loaded via NFS, making any additional kernel and libraries they might have installed available.

The user's resource usage is kept under the same checks they would be for any other usage of g5k. Node reservations for the purpose of running notebooks fall under the same limits as any other node reservation. Front-ends are shared by nature, and limits on resource usage are enforced using `cggroups`. Labs access the user's `homedir` that are limited in size and only accessible by the users themselves.

4.3.2.5.5 Difficulties encountered

jupyter-notebook tooling and jupyter-lab: The `jupyter-notebook` provides a set of subcommands to manipulate running instances of `jupyter-notebooks`, giving users the ability find all running instances, and stop specific instances from any terminal. If the `list` subcommand includes instances of `jupyter-labhub`, the `stop` subcommand does not appear to work in our setup. This is the reason we decided to use a `kill` command to terminate instances of `jupyter-labhub`.

Remote spawners and specification compliance: The structure of **JupyterHub** spawner modules seems to favor local processes, or remote processes under highly controlled monitoring systems. Most notably the fact that the `poll` function is meant to recover the exit status of the lab process might be trivial for local processes, and possible when using docker or a task manager such as slurm or OAR, but becomes complicated for processes open on remote machines started as background tasks under `sudo`. Fortunately, **JupyterHub** doesn't seem to use the exit code in any capacity, and is mostly interested in knowing whether the process is alive or dead.

Hub-side port selection and port deadlock: Another problem with remote spawning is **JupyterHub**'s tendency to select the port to use. The base implementation of the spawner, on which most other implementations depend will select a free port on the hub machine to add as a `port=` argument when building the lab command. Port selection is important since the `start` command must return the port on which the instance is running along with the hostname of the instance. The obvious problem being that a free port on the hub machine might not be free on the front-end.

As is standard, setting the `port=0` argument on the lab command would result in the lab selecting a random free port on the machine on which it is running. And using the `jupyter-notebook list` subcommand it is possible to retrieve the port of an instance whose process-id is known. However trying to use this approach with `jupyter-labhub` leads to a deadlock. To appear in the list subcommand an instance of `jupyter-labhub` must be fully initialized, and initialization requires the lab instance to establish contact with the hub. Parallely the hub is still waiting on the port and hostname information from the `start` command, and will not accept connections from unknown labs. Hence, we find ourselves in a situation where the hub is waiting on the spawner for the port number, the spawner is waiting for the lab to appear in the list, and the lab cannot appear in the list until it has established contact with the hub.

To avoid the deadlock while limiting port conflict, the port selection for front-end instances is performed by the `jupyterctl` script, using `shuf` to generate a list of randomized ports and `netcat` to scan if the ports are available.

4.3.2.6 Conclusions

In this deliverable we presented the integration of Jupyter notebooks to the g5k platform. We showed our analysis of the different use notebooks could have on an experimental computing testbed, how other similar testbeds covered these use cases, and presented how our JupyterHub integration covers these uses.

Discussion with g5k users was enlightening concerning the scope of use notebooks already had on the testbed. We would recommend any testbeds looking to support notebook to survey their users to understand the important uses to cover.

Despite some architectural oddities, JupyterHub has shown itself to be extremely adaptable. The wealth of modules available cover most standard installation needs, and custom modules are relatively easy to build even for uncommon infrastructures like ours.

It is too early to say if notebooks will significantly impact reproducibility of experiments in computer science. Notebooks are not inherently reproducible, and good practices, such as those described here, are necessary to guarantee reproducibility. But the adoption of such good practices can only happen if platforms first lower the entry barrier to notebooks to that of ad hoc scripts.

4.4 GRID'5000 METADATA BUNDLER

When running experiments on Grid'5000, users generate metadata across multiple services. This metadata is useful for reproducibility purposes or scientific dissemination. The g5k-metadata-bundler is a tool designed to retrieve metadata across all the different services and bundle them in a single archive. The bundle only retrieves metadata generated by Grid'5000 services, the collection of data generated by the users experiment is beyond the scope of this application.

This can be found at https://www.grid5000.fr/w/Grid5000_Metadata_Bundler

Usage

G5k-metadata-bundler is installed on every site frontend in Grid'5000. It can only be executed from the site frontends.

```
g5k-metadata-bundler -s SITE -j JOBID [-o NAME]
-v, --version          Print g5k-metadata-bundler version
-s, --job-site SITE    [MANDATORY] Grid'5000 site from which to extract
-j, --job-id JID      [MANDATORY] Job id of the OAR job to extract
-o, --output NAME     Bundle name to use for the directory/archive
```

Users do **not** need to operate the bundler on the same frontend as the site the jobs was executed on. The bundler download all data pertaining to the queried job and bundle in a archive named `g5k-bundle-SITE - JID .tar.gz` or if an output name has been provided `NAME .tar.gz`. The bundle is provided in as a tar.gz archive which can be manipulated by using the following commands:

- Listing

```
tar -tzf NAME .tar.gz
```

lists all files contained within the bundle

- Extraction

```
tar -xzf NAME .tar.gz
```

extracts all files to a directory with the same name as the bundle

Users operating on older versions of Windows might require thrid party software to unpack the bundle. (often 7-zip)

Example usage

```
user@fsophia:~$ g5k-metadata-bundler -s nancy -j 3003030
Running g5k-metadata-bundler for job 3003030 at nancy
Downloading https://api.grid5000.fr/stable/sites/nancy/jobs/3003030
Downloading https://api.grid5000.fr/stable/sites/nancy/clusters/graouilly/nodes/graouilly-1?version=7f6b81c2621c6ed3a4fac632f213436813495755
Downloading https://api.grid5000.fr/stable/?version=7f6b81c2621c6ed3a4fac632f213436813495755&deep=true
Downloading https://api.grid5000.fr/stable/sites/nancy/metrics?job_id=3003030&nodes=graouilly-1
Generating README
Compressing bundle
Bundle created at g5k-bundle-nancy-3003030.tar.gz
user@fsophia:~$ ls -lh g5k-bundle-nancy-3003030.tar.gz
-rw-r--r-- 1 user g5k-users 456K Jul 19 09:50 g5k-bundle-nancy-3003030.tar.gz
user@fsophia:~$ tar -tzf g5k-bundle-nancy-3003030.tar.gz
g5k-bundle-nancy-3003030/
g5k-bundle-nancy-3003030/g5k-oarjob-nancy-3003030.json
g5k-bundle-nancy-3003030/README
g5k-bundle-nancy-3003030/g5k-resource-nancy-graouilly-1-7f6b81c2621c6ed3a4fac632f213436813495755.json
g5k-bundle-nancy-3003030/g5k-monitoring-nancy-graouilly-1-3003030.json
g5k-bundle-nancy-3003030/g5k-refapi-7f6b81c2621c6ed3a4fac632f213436813495755.json
```

Figure 12: Grid'5000 metadata bundler usage and example

D3.6: Developments for the 3rd cycle



The bundle contains these different file types:

- ➔ g5k-oarjob-SITE-JID.json: Job files
Contains the information for a given OAR job JID at Grid'5000 site SITE such as:
 - submission, start, and end dates
 - user and group (group granting access) of the job
 - job types and properties
 - command executed by the job
 - list of resources attributed to the job
 - OAR events for the job

This information is extracted from the jobs API

- ➔ g5k-resource-SITE-NODE-VERSION.json: Resource files
Contains information about a single NODE, such as:
 - Node architecture, bios, ram, and cpu information
 - network, storage, and monitoring devices
 - base configuration information

The bundle will contain one such file for each of the nodes involved in a job.

This information is extracted from the reference API

The VERSION of the information contained in this file will match what it was on the day the job was executed.

- ➔ g5k-refapi-VERSION.json: Reference API files
Contains a full copy of the reference API at VERSION
This can be used to look up information about nodes not directly used by the bundled oar jobs.
The resources files are a subset of this file.
- ➔ g5k-monitoring-SITE-NODE-JID.json: Monitoring files
Contains all the monitoring measurements made by [Kwollect](#) for a NODE during the oar job JID.
The contents will vary depending on how much monitoring was enabled for a given job. See default metrics in [Monitoring Using Kwollect](#).
Often the heaviest files in the bundle
This information is extracted from [Kwollect](#)
- ➔ README: The readme file
Contains information pertaining to the execution of the bundler such as:
 - Bundler version
 - Execution date
 - List of warnings and errors that happened during bundling
 - List of files included in the bundle with a short description

User will also find at the end of every file a small bundler info segment. This segment contains the date at which the file was generated, warnings raised by the file generation and a list of references indicating how this file relates to other files in the bundle.

As the bundler is still in alpha version, we welcome comments and feature requests.

The following is a list of features we are already looking at:

- ➔ Concerning bundle contents
 - Bundling multiple jobs in a single archive
 - Bundling based on job names
 - Better management of monitoring information when it is too big for download
 - Adding information concerning the standard environment to the bundle
 - Adding image deployments to the bundle
 - Adding information concerning the deployed images

- ➔ Concerning bundler operation
 - No-compress mode where the bundle is left as a directory containing all files
 - Appending new files to an existing bundle
 - Reduce memory footprint
 - Assess viability of parallel downloads

4.5 DISTRINET (DELAY BASED FIDELITY MONITORING OF NETWORK EMULATION)

On the Fed4FIRE+ testbeds it is possible to large scale networking experiments, but they are restricted to the physical limits (e.g. only up till 11 network cards on nodes), so it's not infinite scale. Mininet on the other hand is a tool for network simulation but is limited to a single machine. If we use the power of the testbeds with a lot of nodes and deploy a distributed version of mininet on this, that would scale to really huge networks.

Our objective was to enhance network experiment tools in order to address the orchestration of large-scale experiments on grid and cloud environments and make it easier to automatically reproduce experiments. This section introduces Distrinet-HiFi: a Distrinet plug-in to monitor fidelity of emulated experiments based on measurement of packet delays.

This info can be found at <https://github.com/distrinet-hifi>

4.5.1 Prerequisites

The scripts given here use `apssh` and `asynciojobs` to remotely run parallel commands on a number of nodes. First make sure you have a recent version of Python (≥ 3.6), then install those on your computer:

```
pip3 install apssh asynciojobs
```

You also need to have a slice in R2Lab and your computer must be able to log onto the gateway node. If this is not the case already, you can ask to register for an account.

If you would rather use your own cluster of computers to deploy Distrinet-HiFi and/or run the proposed experiments manually, you can ignore this step.

4.5.2 Installation

To set up the experiment, the R2Lab nodes must be correctly configured and running the latest stable version of Distrinet. You can use the already available images to set up your testbed, with one master node and one or more worker nodes:

```
rhubarbe load -i u18.04-distrinet_hifi_leader $LEADER_NODE
rhubarbe load -i u18.04-distrinet_hifi_worker $WORKER_NODE_1
$WORKER_NODE_2 ...
```

You can also manually install the testbed if you do not wish to use R2Lab. Make sure to have a recent Linux Kernel (the tool has been tested on v4.15.0) on all your nodes then install `bcc` and download and install Distrinet. Then copy `hifi.py` to the mininet code directory in your master node (`~/Distrinet/mininet/mininet/`), and the rest of the files to a `~/experiment/` directory you would have created in all the nodes of your testbed.

4.5.3 Usage

First import the Distrinet-HiFi library in your Distrinet script:

```
from mininet.hifi import Monitor
```

and wrap your experiment in the monitoring process:

```
monitor = Monitor(net)
monitor.start()
monitor.wait()
# run your experiment...
monitor.stop()
monitor.receiveData()
monitor.analyse()
```

Before running your experiment, initialise the monitoring agents on each node of your cluster:

```
python3 agent.py --ip=NODE_IP --bastion=LEADER_IP
```

4.6 IMPLEMENTING SFA SUPPORT ON AN ESTABLISHED HPC-FLAVORED TESTBED GRID'5000: LESSONS LEARNED²¹

The Slice-based Federation Architecture (SFA) is the de facto standard framework for managing testbeds, federations of testbeds, and users access to these federations. It is the foundation of most major testbed federations in the world, including GENI and Fed4FIRE. However, there remain some testbeds that were designed and grew outside of this world, making different, interesting and sometimes better design choices. In this section, we describe how we added support for the GENI Aggregate Manager to the Grid'5000 testbed, a major testbed focused on HPC and Cloud that was developed for the most part independently for the last 15 years. From this experience, we draw some lessons and recommendations that could help improve the testbed management ecosystem.

4.6.1 Introduction

As the internet gained widespread adoption in the late 1990s, new types of distributed systems, such as peer-to-peer networks and content distribution networks, began to emerge. For computer scientists, studying and experimenting was a complicated process, which often required borrowing access to machines in other institutions and dealing with heterogeneous configurations and software. Faced with these difficulties, the community moved to organize shared testbeds.

These testbeds would be centrally managed and offer a more consistent method to access resources over multiple sites, on which experiment isolation would be achieved using techniques such as network slicing, virtualization, or bare metal loading. The major examples of such community testbeds were Emulab²² focused on network experimentations using emulation, PlanetLab²³ focused on highly distributed applications, ORBIT²⁴ for wireless networks, and DETER²⁵ focused on security. These testbeds quickly became an essential part of the study of distributed systems.

Recognizing the importance of such testbeds, the US National Science Foundation (NSF) contributed to setting up the GENI Project Office to develop the next generation of testbeds. By setting it up as a federation of testbeds, GENI^{26,27} would be able to provide a wide array of different resources over a large number of physical locations through a common interface.

In the GENI architecture, three parties interact: the user, the federation, and the testbeds. The federation provides user authentication and authorization. The federation *Clearinghouse* issues user certificates and signed credentials which authorize users to interact with federation-wide namespaces called *slices*. Using these certificates and credentials, users can contact the different testbeds through an am to add resources to their slice. Resources are provided as part of testbed-wide namespaces called *slivers* which are added to the user slice. This setup delegates user and experiment management to the federation, letting ams decide which federation's credentials to accept. GENI proposes a set of standards for the different elements of this *Slice-based Federation Architecture* (SFA). The current version of the AM API is AMv3.

Today, GENI APIs are widely seen as the de facto standard for testbed management and are used by other testbed federations such as Fed4FIRE in Europe. However, other testbeds have been designed and developed outside of

²¹ This can also be found at <https://hal.inria.fr/hal-02962845>

²² B. White, J. Lepreau, L. Stoller, R. Ricci, et al., "An integrated experimental environment for distributed systems and networks," in OSDI, 2002

²³ B. N. Chun, D. E. Culler, T. Roscoe, A. C. Bavier, et al., "Planetlab: An overlay testbed for broad-coverage services," Computer Communication Review, 2003.

²⁴ D. Raychaudhuri, M. Ott, and I. Seskar, "ORBIT radio grid tested for evaluation of next-generation wireless network protocols," in TRIDENTCOM, 2005

²⁵ T. Benzel, R. Braden, D. Kim, C. Neuman, et al., "Experience with deter: A testbed for security research," in TRIDENTCOM, 2006.

²⁶ M. Berman, J. S. Chase, L. Landweber, A. Nakao, et al., "Geni: A federated testbed for innovative network experiments," Computer Networks, vol. 61, 2014.

²⁷ R. McGeer, M. Berman, C. Elliott, and R. Ricci, The GENI book. Springer, 2016.

the SFA world. In this section, we discuss the challenges we faced, and the lessons we learned, implementing a GENI AM in an established testbed: Grid'5000. We hope that this discussion will be useful to testbed designers and operators, to understand what implementing an AM encompasses. We also hope that it will be useful to identify future SFA evolutions.

This section is structured as follows. Section 4.6.2 provides some background on the g5k testbed. Section 4.6.3 offers a comparison of GENI and g5k APIs and presents g5k's implementation of an Aggregate Manager. In section 4.6.4 we focus on how the differences between GENI and g5k affect our work, and how we would hope GENI and g5k could change as a result. Finally, we discuss other testbed control frameworks in section 0 before concluding.

4.6.2 Background on Grid'5000

The g5k project²⁸ was initiated in 2003 by the French HPC (High Performance Computing) research community. At the time, the focus of this research community was on Grid computing, and there was a clear need for a large scale highly reconfigurable testbed in order to experiment at scale. Many of Grid'5000's design decisions are inherited from that time and inspired from the HPC world, for example like most nationwide computing grids, it is distributed over several sites, and it uses an HPC resource manager (batch scheduler) to manage resources.

Over the years, the focus of the testbed has evolved, but remained rooted in the French HPC and distributed systems community. Its focus today is similar to testbeds such as Chameleon²⁹ or CloudLab³⁰, covering topics like HPC, AI, Clouds, Edge Computing, Big Data. The services it offers are similar to those offered by the aforementioned testbeds³¹: bare metal provisioning, network isolation, monitoring services, etc. It also offers similar hardware: x86 and ARM servers, HPC networks (e.g. InfiniBand, Omni-Path), GPUs, etc.

Since 2017 in the context of the EU H2020 Fed4FIRE+ project, g5k joined the Fed4FIRE testbed federation, had to support SFA, and thus implement and operate an AM. This raised a number of challenges, detailed in the next section.

4.6.3 Implementing the Slice-based Federation Architecture in g5k

4.6.3.1 Technological Overview

4.6.3.1.1 GENI AMv3 API

This API is based on XML-RPC over HTTPS. Calls are made using an SSL client certificate. This user certificate is issued by the federation and is used by the ams to filter access to resources. Additionally, for most operations, users will provide a *credential* in the call parameters (an XML snippet signed by the federation *Clearinghouse*). The credential describes the permissions given to the subject of the client certificate (the user) over the target slice. Using the API, users are able to add am-local *slivers*, representing testbed resources, to their federation-wide *slice*.

In a typical use case, the user would first query the AM for its capabilities and available resources, presented in an *advertisement RSpec*, using the **getVersion** and **listResources** API calls respectively. To claim resources for a slice, the user sends a *request RSpec* in a **allocate** call to every involved am. The ams book the requested resources in corresponding slivers and answers the **allocate** call with a list of slivers and a *manifest RSpec* describing the booked resources. The user must then confirm their booking using a **provision** and a

²⁸ D. Balouek et al., "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, 2013, pp. 3–20.

²⁹ K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, et al., "Chameleon: A scalable production testbed for computer science research," in *Contemporary High Performance Computing*, CRC Press, 2019

³⁰ R. Ricci, E. Eide, and C. Team, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *login*, vol. 39, no. 6, 2014.

³¹ L. Nussbaum, "Testbeds Support for Reproducible Research," in *ACM SIGCOMM 2017 Reproducibility Workshop*, Los Angeles, United States, 2017

performOperationalAction call, during which the ams will start and configure the booked resources. At any time, the user can use **status** to find the logical and operational state of the chosen slivers, and **describe** to obtain a manifest of chosen slivers. Once an am shows a provisioned sliver in the *ready* state, the user can connect to the underlying resources to perform their experiment. Once finished, the **delete** call is used to release the resources booked in the chosen sliver.

RSpecs are XML snippets used by the AM API to describe resources, and come in three flavors: *advertisement*, *request*, and *manifest*. RSpecs contain `<node>` and `<link>` elements, representing respectively computational resources and network links. Nodes are advertised with a `component_manager_id` (the testbed in charge of the node), a unique identifier for the node (`component_id`), its type (bare metal, virtualized, ...), disk images available, and possibly hardware types and network interfaces. In a *request RSpec*, the user must only specify a label, the `client_id`, the component manager handled by the AM where the node will be located using the `component_manager_id`, and the node type. Additionally, users can specify an image to deploy, a hardware type or a specific node. *Manifest RSpecs* mostly copy the content of the corresponding *request RSpec* completing it with the identifiers of the allocated resources and additional login information.

4.6.3.1.2 g5k

For most use cases, g5k users do not access the testbed directly through the API. Users usually connect via SSH to the sites' front-ends and use available commands to manage resources. g5k's testbed control infrastructure is based on OAR³² (an HPC resource manager) and related tools. Each g5k site runs an independent instance of OAR. Users looking for specific resources are invited to use the g5k website³³ to find the clusters (sets of homogeneous nodes) available at different sites.

The granularity of resource bookings can go from a whole cluster to a single CPU-core. Sub-node level bookings rely on Linux *cgroup/cpuset* mechanisms to maintain isolation between jobs. OAR also allows for the building of complex requests based on resource hierarchies. A user can choose to book ten CPU cores using a `core=10` request or can request specific topologies such as having one core on two separate CPUs of five different nodes all belonging to a single cluster using `cluster=1/host=5/cpu=2/core=1`. These hierarchies can further be constrained through property filtering, to restrict execution on specific cluster nodes, or to filter resources based on intrinsic properties such as memory size or network bandwidth. Property filtering relies on an SQL request allowing users to write complex constraints. A single booking can contain multiple resource hierarchies each with their own property constraints. Furthermore, users should specify a job walltime (duration), optionally a command to execute, and can specify a job type. Job types can be used to request the default system image (avoiding bare metal deployment) or the ability to deploy disk images, which is performed using a second tool, KaDeploy³⁴. Resources are automatically released after the walltime is reached, or if the provided command finishes, or if the job is forcefully terminated using OAR.

g5k offers a REST API, grouping the different services. Part of this API, called the *reference API*, offers a functionality similar to SFA's `listResource` by providing information on available resources, using `HTTP GET` on `/sites/<site>/clusters/<cluster>/nodes/<node>`. The *jobs API* allows users to create and manage OAR jobs, fulfilling the function of the AMv3 `allocate` and `provision` calls. Jobs are created using HTTP POST requests specifying walltimes, commands, resource hierarchies, properties, and types. Following the REST principles, these operations create a new endpoint for the jobs that users can query to track the state of the OAR job or DELETE to end the job. Conversely the *deployments API* endpoints can create and monitor KaDeploy's installation of disk images, fulfilling part of the `provision` call. The API also offers endpoints to alter network topology, monitor power usage, and access account management.

³² N. Capit, G. Da Costa, Y. Georgiou, G. Huard, et al., "A batch scheduler with high level components," in IEEE CCGrid 2005, 2005.

³³ <https://www.grid5000.fr/w/Hardware>

³⁴ E. Jeanvoine, L. Sarzyniec, and L. Nussbaum, "Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters," USENIX ;login:, vol. 38, 2013.

4.6.3.2 Design and Implementation of an AM for g5k

The *geni-tools* suite³⁵ provides a base implementation of an aggregate manager, providing parsing XML-RPC requests, structuring responses, and management of certificates, credentials, slices, and slivers. Table 10 shows what calls are made to the g5k API for a specific GENI AM call. The AM is authenticated with an SSL certificate allowing it to impersonate g5k users during calls.

getVersion	No Grid'5000 API calls
listResources	GET /sites/<site>/status GET /sites/<site>/clusters/<cluster>/nodes/<node> GET /sites/<site>/internal/kadeployapi/environments
allocate*	POST /sites/<site>/jobs
provision*	POST /sites/<site>/deployments/
status & describe*	GET /sites/<site>/jobs/<job_id> GET /sites/<site>/deployments/<id>
delete*	DELETE /sites/<site>/jobs/<job_id>
performOperationalAction	No Grid'5000 API calls
Renew	Not currently implemented
*: these calls require user impersonation. Users are found/created using: GET /users/engines/fed4fire/external_id	

Table 10: Grid'5000 API calls triggered by AMv3 API calls

4.6.3.2.1 Access Management

The AM is configured to accept any calls passed with a valid certificate signed by the Fed4FIRE federation. However, most of our internal tooling relies on the existence of local user accounts. We therefore had to implement the dynamic creation of Grid'5000 user accounts for federation users, and also the mapping of existing Grid'5000 user accounts to federation user accounts for requests coming through the AM.

Our user management service (UMS) has been updated to add the possibility for accounts to have external identifiers, allowing us to add a *Fed4FIRE user URN* (Uniform Resource Name) to accounts. The UMS was also extended to provide a method for retrieving an account associated with an URN and, if none exists, creating new a account on the fly using the information provided in the user certificate. Accounts created only have a username, the email address, and an SSH public key, all taken from the certificate key. Such accounts are only valid for a month, and extending the validity requires users to provide information about their identity and their affiliation. With this setup, new users coming from Fed4FIRE can immediately use the testbed, while giving us visibility on more long-term users.

4.6.3.2.2 Listing Resources

The AM builds the *advertisement RSpec* by interrogating the *reference API*. Since this information is public (as it is available on the g5k wiki), this call only checks for a valid user certificate and does not check whether the caller has a valid g5k account.

The am advertises every node in the testbed. The site to which a node belongs is exposed as part of the *component_manager_id*, the cluster is exposed as the *hardware_type*, and the node URN is built from the corresponding *reference API* path. OAR properties other than the cluster are not exposed through the *advertisement RSpec*. For each node, the list of current g5k managed disk images is provided.

³⁵ <https://github.com/GENI-NSF/geni-tools>

4.6.3.2.3 Allocation and Provisioning

When faced with an **allocate** request, the AM will first check the validity of the user certificate and the credentials provided by the federation, and find the corresponding g5k user account or create one. The *request RSpec* is then parsed for relevant node requests. These nodes are then started and added to the user slice. Lastly, if OAR manages to book all the requested resources for immediate use, the corresponding manifest is generated and sent back to the user, otherwise the allocation fails.

During allocation, each node requested from g5k is started using a single-node OAR job. These jobs are then turned into slivers added to the user slice. Nodes requesting a specific disk image will be booked using the OAR **deploy** type, whereas other nodes will use the **allow_classic_ssh** type (providing the default system environment, without bare metal deployment). The walltime is set to the requested end date or the end of the user credential validity. Internally, the AM sets the expiry of the slivers at 10 minutes.

During a **provision** call, the AM extends the validity of the provisioned slivers to the end of the walltime of the OAR jobs, and deploys requested disk images to the provisioned nodes.

4.6.3.2.4 Node Access

Once their nodes reach the **geni_ready** state, users will be able to connect to them using SSH. Nodes started without a specific disk image allow users to connect to their personal account on the node and access their NFS home directory. On nodes started with a specific disk image, users will connect to the root account. In all cases users will need to first connect to one of the g5k SSH access gateways, and the connection will use their Fed4FIRE user certificate's private key as identity. The information on how to connect to the gateway and to the node is available in the *manifest RSpec* produced by the **allocate**, **provision**, and **describe** API calls.

4.6.3.2.5 Deletion

If no provisioning has been performed in the ten minutes following an allocation, the allocated slivers are marked for expiry. Expired slivers are deleted by the AM, stopping the underlying OAR job and freeing resources. Moreover, resources can be freed using the **delete** API call.

4.6.3.3 Current Status

After the implementation of all of the above, it is now possible to reserve and provision Grid'5000 resources through Fed4FIRE+'s standard GUI tool (jFed³⁶). Additionally, support for *Experiment Specification (ESpec)*, a Fed4FIRE-built solution for experiment packaging, also works immediately. The AM necessitated 2154 new lines of Python code on top of the code provided by *geni-tools*, and makes use of an external 1215 lines from the *python-grid5000* library.

4.6.4 Divergence between GENI and g5k

In this section we discuss the points of contention encountered while implementing GENI AM on g5k, and how they relate to the difference in objectives between GENI and g5k. Where relevant, we also make recommendations.

4.6.4.1 Support for Multi-Site Testbeds

g5k is a single testbed distributed on multiple sites. Every site is managed by the same technical team and user accounts are centrally managed, but resource bookings happen on a site-by-site basis. The GENI AM API is designed to allow AMs to aggregate multiple testbeds, each identified by their **component_manager_id**. Since most testbeds run their own AM, tools such as jFed have poor support of AM with multiple component

³⁶ <https://jfed.ilabt.imec.be/>

managers. However, giving each site its own AM would turn internal links within g5k into AM links. Using a single g5k AM is thus more consistent with g5k's internal structure.

We have therefore chosen to stay with the initial design and expose each g5k site as a component manager of the global g5k AM. Poor support for that design is mitigated by the fact that site information is only necessary when users do not request a specific node or cluster. Our implementation overrides component manager information to always match the requested resource. In the event the request does not include enough information to derive a site, we arbitrarily made our AM default to our largest site. This keeps g5k usable for end-users even when used through tools with incomplete support for the component manager concept.

4.6.4.2 RSpec Limitations for Resources Description

Because of GENI's focus on networking, RSpec documents used to describe and request resources are centered around two main elements: nodes and links. Nodes are considered as individual computational elements in the middle of the network being tested. This is a striking departure from the g5k setup where the cluster a node belongs to is a major characteristic of the node.

4.6.4.2.1 Resource Description

Advertisement RSpecs describe nodes using a unique identifier, the identifier of the AM they belong to, one or more types of bookings (bare-metal, virtualized, ...), and possibly a list of available disk images. Specification of the actual underlying hardware was added later with the introduction of hardware types. Hardware types are sub-elements added to a node. Depending on the testbed, hardware types can be used to describe the node's make and model (e.g. Emulab's 'nuc8650'), the underlying architecture (e.g. Emulab's 'x86_64'), a generic type (e.g. w-iLab.t's 'LTE-FEMTOCELL'), or a cluster (as in g5k).

The problems with the hardware type mechanism are twofold. First, a simple hardware type cannot convey the full configuration of a node, forcing users to reference external information alongside the advertisement RSpec. To remedy this problem Fed4FIRE has designed the `hwinfo37` RSpec extension, which allows AMs to add hardware type descriptions to the advertisement RSpec. Second, many interesting properties of a node, such as CPU speed or RAM size, are quantitative. This is at odds with the hardware type system of assigning a tag to the node. This use case appears to be filled in part by Emulab's `fd` elements which allows one to specify a value to a name property. However, both `hw-info` and `fd` are not part of the mainline standard, and their availability varies greatly.

Furthermore, RSpec's focus on a node as an independent unit of computing creates an inefficient organization of the resource description. Disk images, which are not intrinsic properties of nodes, appear as sub-elements of the nodes. This means that a disk image element is repeated on every node it can run on. For g5k, where provided images can run on every available node, this means each disk image is repeated 828 times in a single *advertisement RSpec*.

Comparatively, g5k's *reference API*³⁸ can provide a node's cluster, CPU (architecture, microarchitecture, speed, caches sizes), RAM size, network adapters (type, addresses, connected switches), bios version, and much more information, in a more compact and human readable JSON format³⁹. Moreover, most of this information can also be used to filter resources in OAR jobs.

³⁷ <https://grid5000.gitlabpages.inria.fr/gcf-grid5000-plugin/hwinfo.html>

³⁸ <https://public-api.grid5000.fr/stable/sites/nancy/clusters/gros/nodes.json?pretty=1>

³⁹ D. Margery, E. Morel, L. Nussbaum, et al., "Resources Description, Selection, Reservation and Verification on a Large-scale Testbed," in TRIDENTCOM, 2014

4.6.4.2.2 Resource Booking

The difficulties with describing resources become a problem when the time comes to request resources. *Request RSpecs* work well in cases where users want to book a specific node or in cases where any node within a testbed would do. However, users interested in nodes based on a set of properties or on relations to other nodes will face more difficulties. Hardware types can be used for requests on some testbeds, but Fed4FIRE+ tools using them only allow one type per node. This makes hardware types suitable to describe the node's complete configuration, but not to be used as a set of piecemeal properties to combine. In g5k, this means hardware types are most suited to indicate the node's cluster, since all nodes in a cluster have the same hardware configuration. Hierarchies and properties as found in g5k, where users can ask for a specific number of nodes distributed over a given number of clusters, fulfilling a complex set of properties using logic operators, cannot be expressed in RSpec. Users are forced to choose the clusters themselves, a decision best handled by the g5k's resource scheduler. These constraints are prejudicial to GENI APIs, because the ability to obtain resources based on properties or relations helps push experimental reproducibility beyond a simple replication on the exact same hardware, which is an important part of experimentation with distributed systems.

Recommendations:

- ➔ Explore standardization of resources description in RSpecs.
- ➔ Explore standardization of methods for building the *Request RSpec* from the *Advertisement RSpec*.

4.6.4.3 Allocation and Provisioning

4.6.4.3.1 Two-step request

In the AMv3 API, starting a new node is a two-step process. First, the allocate is made with the request RSpec. The AM schedules resources and creates corresponding slivers before informing the user. The requested resources are held for a short time and released should the user fail to provision the slivers. AMs are told to implement allocate as a quick, cheap, and readily undoable operation. Second, users are expected to provision the wanted slivers. The resources in the provisioned slivers are supposed to be started at this point. Some testbeds also require the `geni_start` command to be issued using the `performOperationalAction` call. This multi-step process is useful in a testbed federation setup, as the allocation allows a user to test the availability of resources across all the testbeds involved in the experiment with provisioning only happening if a sufficient number of resources is available, and canceling if not. However, this setup's efficiency is dependent on whether a testbed can implement `allocate` as a lightweight scheduling. In our current OAR installation, we can schedule resources for future use, but we cannot schedule resources for immediate use without starting them. This makes the allocation process in g5k heavier than expected by the GENI AM API, but in most cases, it allows us to start image deployment immediately when users `provision` the slivers.

4.6.4.3.2 Slice, Slivers, and Jobs

Slices are federation-wide namespaces attributed by the federation *Clearinghouse* to a user, or group of users, for them to run their experiment. With a credential for this slice, the users can ask any AM in the federation to `allocate` resources to the slice. Resources are provided as AM-wide slivers. In g5k, OAR *jobs* are the finest granularity of resources manipulation. Grouping resources into a single *job* is possible but requires complex RSpec transformations and would bind the corresponding slivers together. Giving each node their own OAR *job* makes the RSpec transformation easier and offers a direct link between sliver status and *job* status. However, although OAR can easily perform multiple single-node jobs, the disk image deployment system does not scale as well and is much more efficient when grouping similar deployments in a single operation. This seems to be a recurrent problem since the AM API offers options to force users to `provision` all slivers in a slice at once.

Conspicuously absent from the AM API is a way to make advance reservations. Some testbeds such as WiLab and BonFIRE accept `geni_start_time` options to an `allocate` call, but this capability is far from standard and there is no information on how such advance reservations should be provisioned. Advance reservations guarantee resource availability for large-scale experiments. The testbeds with advanced reservation capabilities, such as

D3.6: Developments for the 3rd cycle



CloudLab⁴⁰ and g5k, allow users to perform preparatory work using smaller allocations before doing their large advance reservation for their experiment.

Recommendation:

- ➔ Improve support for advance reservations.

4.6.4.4 Resource access

To connect to g5k, users first need to SSH into one of two access gateways. From there they are able to SSH into any site front-end or any node they have booked. The SSH keys used in these connections are managed by g5k's user management system. RSpec provides a method to describe how to connect to a node, which is complemented by the proxy extension designed by Fed4FIRE. Since not every testbed provides users with a home directory shared between the nodes, users might rely on this SSH access to setup and execute their experiment. The **provision** call offers the possibility to inject SSH keys into the provisioned nodes to share access with tools and other experimenters. This works in conjunction with the federation *Clearinghouse's* ability to issue credentials for a slice to multiple users to create true experiment sharing between researchers. However, g5k is not built in a manner conducive to this kind of experiment sharing and the option to inject SSH keys poses multiple security problems. First, key injection only works in cases where users deploy a disk image, since KaDeploy is the tool setting the new SSH keys to the root account of these images. In cases where users don't deploy an image, they log on using their personal account to the standard g5k environment and so can only use SSH keys set in their account. Second, even in cases where a key is deployed to a node, connecting into the access gateway requires a g5k account and can only be done using the keys registered in the user account. Third, in cases where the initial user shares his node with a second user who has a g5k account, the second user gains full access to the initial user's home directory. Overall, the mechanisms for experiment sharing and SSH key injection do not seem well suited to testbeds that provide per-user storage accessible from reserved nodes or rely on SSH access gateways for external access.

4.6.4.5 State of the Standard

GENI AMv3 API was finalized in May 2012, and some technological choices, such XML-RPC, have since then lost in popularity in favor of alternatives like REST and JSON. The GENI Engineering Conferences have been considering changes, but no version 4 has yet been proposed. Meanwhile testbeds and federations have worked around the standard by creating RSpec extensions leading to a greater fragmentation of the API.

Recommendations:

- ➔ Resume standardization efforts, consolidate extensions.
- ➔ Explore more recent technological foundations (REST).

⁴⁰ D. Duplyakin, R. Ricci, A. Maricq, G. Wong, et al., "The design and operation of cloudlab," in USENIX Annual Technical Conference, 2019.

4.6.5 Related Work

With SFA being the de facto standard for testbed federation management it pushes testbeds to either adopt GENI stacks or, as g5k did, adopt a compatibility layer to offer a GENI API. This is clearly visible in testbed federations. One could argue that this leads to some ossification in the context of testbed control interfaces, as shown by the limited number of alternatives.

Some testbeds still expose different control interfaces – some of which were created during the early exploratory phase of the GENI project. The ExoGENI subproject built a distributed iaas testbed where resource control was operated by an out of the box cloud management system on top of which a testbed management system based on orca⁴¹ was deployed. In orca APIs, resources are described with extensive sets of properties, and brokers could be used as intermediaries between users and ORCA AMs. ExoGeni testbeds also provide GENI AMv2 API. The ORBIT testbed concurrently developed the omf offering not only an interface for resource management but the management and monitoring of whole experiments. In omf⁴² a resource request can be performed using a programmed description. This concept is also used in Cloudlab although the programmed description generates a standard *request RSpec*.

Other testbed management stacks are more recent. Tsumiki⁴³ is a testbed management system framework in which operators are invited to build testbeds suiting their needs. WalT⁴⁴ is designed to work with cheap hardware therefore allowing any user the ability to easily reproduce both the testbed and the experiment running on it.

The ChameleonCloud testbed leverages the iaas control software, Openstack, to manage its resources, and adds additional tools and services. This allows users to book resources using the standard Openstack clients. Similarly, EdgeNet⁴⁵ is a nascent testbed for geographically distributed experiments, in the vein of PlanetLab, leveraging the Kubernetes container orchestration software.

4.6.6 Conclusions

In this article we presented the implications of implementing the GENI AMv3 API in the g5k testbed. We draw several lessons and recommendations from this work and a comparison of Grid'5000 and GENI capabilities. The main area of improvement we perceive in GENI APIs is the description and selection of resources, which lacks support for advanced filtering or more abstract nodes relationships. Research on this matter is already ongoing and other teams participating in the Fed4FIRE+ project are looking to build a RSpec extension based on ontologies.

A popular direction for recent testbeds is to build their services on industry-grade software like OpenStack and Kubernetes. It will be interesting to understand how this affects, or not, the ability to integrate with a standard API, or how lessons from those widely adopted infrastructure management frameworks could influence the testbed community. Having an interface that can represent the resources and topologies needed for an experiment in a high-level and precise fashion and that can provision and orchestrate these resources anywhere in the world would allow for unprecedented levels for experiment reproduction at a time when computer science is facing a reproducibility crisis.

⁴¹ J. S. Chase and I. Baldin, "A retrospective on ORCA: open resource control architecture," in The GENI Book, R. McGeer, M. Berman, et al., Eds., 2016.

⁴² T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "OMF: a control and management framework for networking testbeds," Operating Systems Review, 2009.

⁴³ J. Cappos, Y. Zhuang, A. Rafetseder, and I. Beschastnikh, "Tsumiki: A meta-platform for building your own testbed," IEEE Trans. Parallel Distrib. Syst., 2018.

⁴⁴ P. Brunisholz, E. Duple, F. Rousseau, and A. Duda, "Walt: A reproducible testbed for reproducible network experiments," in CNERT, 2016.

⁴⁵ <https://edge-net.org/>

5 CENTRAL BROKER

In cycle 1 and 2, the Central Broker was developed as an overarching service that can be utilized by the experimenters to discover resources that span the federation and fulfil their experimentation requirements.

In cycle 2, we developed a testbed selector for the new portal (<https://portal.fed4fire.eu/explore/discover>) which helps in finding the right testbed for an experimenter based on high-level properties of the testbeds.

A nice addition to that would be the combination of the testbed selector with the detailed resource overview of the central broker. Also if experimenters are searching for specific resources (e.g. specific cpu, gpu, wireless card,...) they can find the testbed that serves those resources.

The Central Broker has been adapted and integrated to the new portal of Fed4FIRE+, through which experimenters could discover which testbeds offer the experimental capabilities and technologies they look for. To this end, Central Broker acts as a backend system for the new frontend, making sure that it provides the latest information of the federated testbeds. The new functionalities implemented make use of the Federation Monitor (FedMon, (<https://fedmon.fed4fire.eu>)) service in order to retrieve the latest RSpecs of the federated testbeds and update the inventory with their unique characteristics related to communication technologies and performance capabilities like CPU, RAM and storage.

In the current Fed4FIRE+ architecture, Central Broker belongs in the federation services layer as it is a global service for the federation, similarly to the FedMon service. According to the architecture Figure 13, Central Broker exposes a REST API, which is used by the new portal of Fed4FIRE+, while it leverages the REST API of FedMon to collect the existing RSpec information that FedMon is periodically collecting through the SFA API of the testbeds' Aggregate Managers.

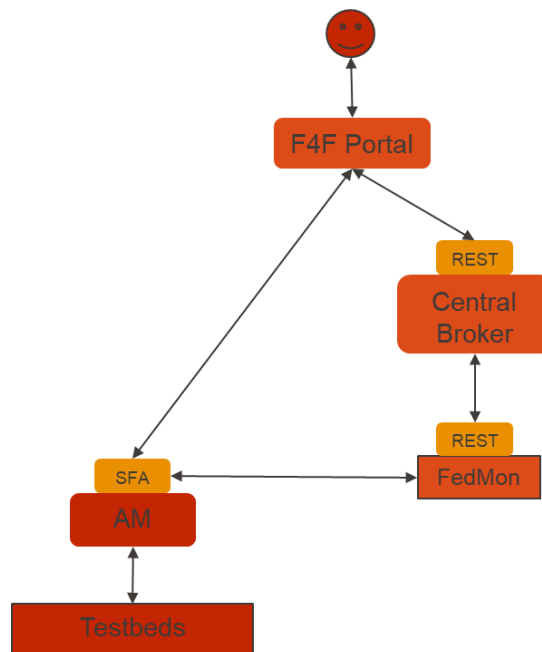


Figure 13: Central Broker Architecture

The REST API of the Central Broker provides a filtering mechanism for resource discovery that can be used by experimenters in the F4F portal in order to identify available resources based on their requirements. The example which can be seen in Figure 14 is a JSON message, which asks the Central Broker to retrieve all the testbeds with resources that meet the following criteria:

- Testbeds that have wireless mobile resources with abgn wireless interfaces and testbeds that have resources with ethernet interfaces with 1Gbps and/or 6Gbps and/or 20Gbps capabilities.

Part of the response from the Central Broker can be seen in the same figure and includes how many resources of which type and features each testbed provides. Specifically in this example there are 15 resources in w-Lab.t testbed that are mobile and have abgn and ac wireless interfaces (see Figure 15 below how this looks in the portal). In the next entry, there is a list of resources that meet the wired capabilities requirements starting with 47 resources in Grid5000 testbed that have 1Gbps ethernet interfaces.

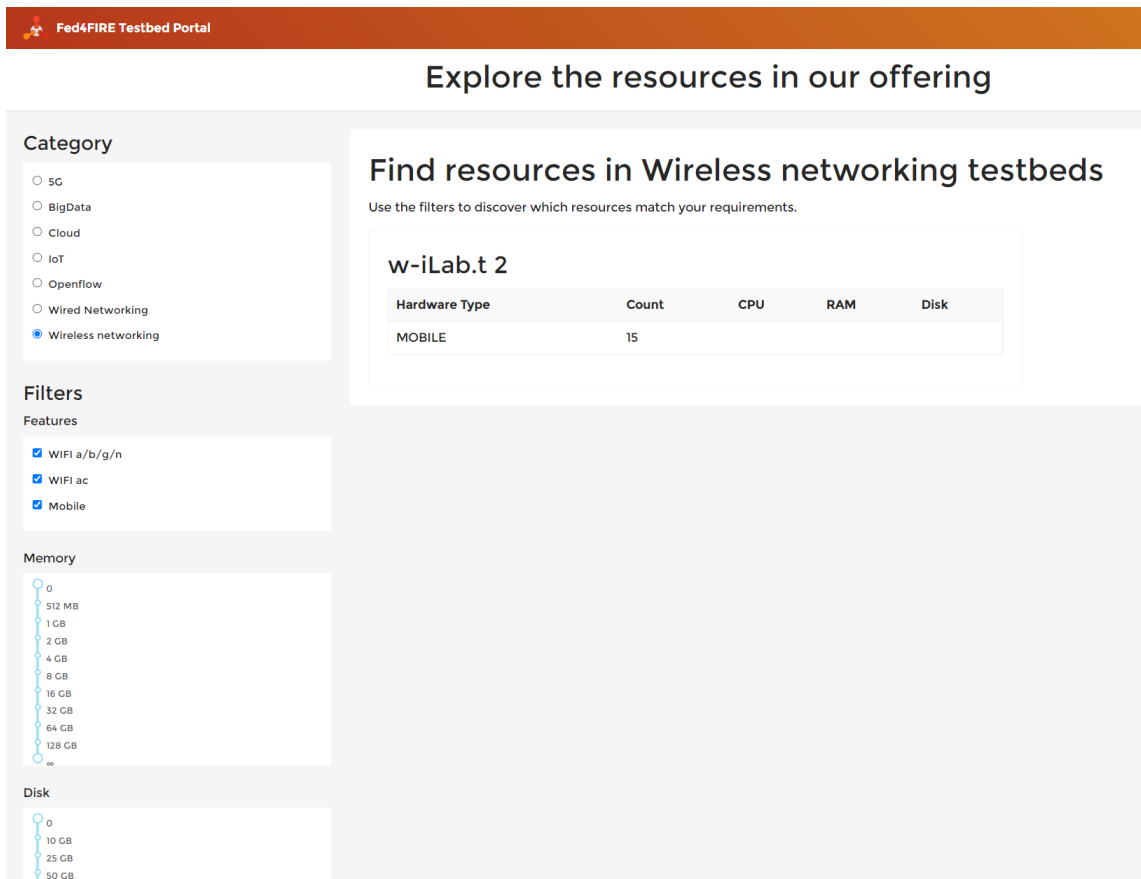


```

1 {
2   "wireless": [{
3     "features": [
4       "mobile",
5       "abgn"
6     ],
7     "resources": [{
8       "count": 15,
9       "hardware_type": "MOBILE",
10      "domain": "wilab2.ilabt.iminds.be",
11      "component_manager": "urn:publicid:IDN+wilab2.ilabt.iminds.be+authority+cm",
12      "cpu_ghz": null,
13      "ram_gb": null,
14      "disk_gb": null,
15      "features": ["mobile", "abgn", "ac"]
16    }]
17  }],
18  "wired": [{
19    "features": [
20      "1gbps"
21    ],
22    "resources": [{
23      "count": 47,
24      "hardware_type": "grcinq-nancy",
25      "domain": "am.grid5000.fr",
26      "component_manager": "urn:publicid:IDN+am.grid5000.fr+edge+authority+am",

```

Figure 14: Filtering based on resource technology and features.



Fed4FIRE Testbed Portal

Explore the resources in our offering

Find resources in Wireless networking testbeds

Use the filters to discover which resources match your requirements.

w-iLab.t 2

Hardware Type	Count	CPU	RAM	Disk
MOBILE	15			

Category

- 5G
- BigData
- Cloud
- IoT
- Openflow
- Wired Networking
- Wireless networking

Filters

Features

- WIFI a/b/g/n
- WIFI ac
- Mobile

Memory

- 0
- 512 MB
- 1 GB
- 2 GB
- 4 GB
- 8 GB
- 16 GB
- 32 GB
- 64 GB
- 128 GB
- ∞

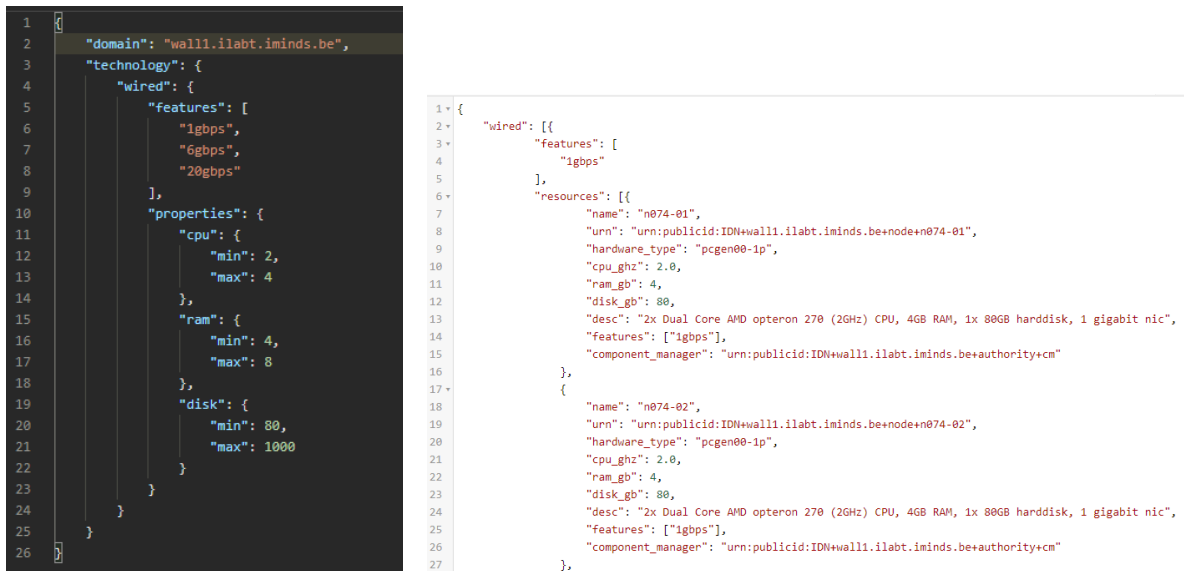
Disk

- 0
- 10 GB
- 25 GB
- 50 GB

Figure 15: Portal view of resource selection (wireless mobile nodes with wifi a/b/g/n/ac)

Once the users get an overview of which testbeds offer resources that meet their requirements in terms of technologies and features, they can proceed by identifying specific resources per testbed that are currently available for reservation and filter them based on CPU, RAM and storage capabilities. The example which can be seen in Figure 16 is a JSON message, which asks the Central Broker to return all the available resources of w-lab.t that meet the following criteria:

- ➔ Resources with at least one of the following wired interfaces “1Gbps, 6Gbps, 20Gbps”, CPU between 2-4GHz, RAM between 4-8GB and storage between 80-100GB.



```

1  {
2  "domain": "wall1.ilabt.iminds.be",
3  "technology": {
4    "wired": {
5      "features": [
6        "1gbps",
7        "6gbps",
8        "20gbps"
9      ],
10     "properties": {
11       "cpu": {
12         "min": 2,
13         "max": 4
14       },
15       "ram": {
16         "min": 4,
17         "max": 8
18       },
19       "disk": {
20         "min": 80,
21         "max": 1000
22       }
23     }
24   }
25 }
26
27 {
28   "wired": [
29     {
30       "features": [
31         "1gbps"
32       ],
33       "resources": [
34         {
35           "name": "n074-01",
36           "urn": "urn:publicid:IDN+wall1.ilabt.iminds.be+node+n074-01",
37           "hardware_type": "pcgen00-1p",
38           "cpu_ghz": 2.0,
39           "ram_gb": 4,
40           "disk_gb": 80,
41           "desc": "2x Dual Core AMD opteron 270 (2GHz) CPU, 4GB RAM, 1x 80GB harddisk, 1 gigabit nic",
42           "features": ["1gbps"],
43           "component_manager": "urn:publicid:IDN+wall1.ilabt.iminds.be+authority+cm"
44         },
45         {
46           "name": "n074-02",
47           "urn": "urn:publicid:IDN+wall1.ilabt.iminds.be+node+n074-02",
48           "hardware_type": "pcgen00-1p",
49           "cpu_ghz": 2.0,
50           "ram_gb": 4,
51           "disk_gb": 80,
52           "desc": "2x Dual Core AMD opteron 270 (2GHz) CPU, 4GB RAM, 1x 80GB harddisk, 1 gigabit nic",
53           "features": ["1gbps"],
54           "component_manager": "urn:publicid:IDN+wall1.ilabt.iminds.be+authority+cm"
55         }
56       ]
57     }
58   ]
59 }

```

Figure 16: Resource filtering based on testbed and hardware capabilities.

Part of the response from the Central Broker can be seen in the same figure, which provides a detailed list of the available resources in w-Lab.t testbed with specific resource descriptions and resource IDs that meet the requirements of the user and can be used by jFed to reserve these resources and start experimenting.

In order for the Central Broker to provide the aforementioned capabilities for the new Fed4FIRE+ portal, it leverages the collected RSpecs by FedMon together with information that each of the testbed has provided in their corresponding documentation pages regarding the hardware capabilities of their resources. This static information has been collected and stored in the Central Broker’s database and linked with the “hardware_type” (https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/rspec.html#_element_code_hardware_type_code) field that every testbed advertises in their RSpecs through the SFA API.

6 RESOURCE RECOMMENDATION SERVICE (RRS)

The resource recommendation service (RRS) is going to be offered to end users to help them to select the most appropriate resources (across the available FED4FIRE+ testbeds) for realizing an experiment, while exploiting its capabilities at full potential using the federation as a whole (and not in a fragmented manner through the use of specific testbeds). Specifically, the development of the RRS aims to a) increase functionality of the federated testbeds, and b) improve the user experience, while also contributing to a) increasing visibility of the federation, and b) supporting a sustainable solution for the federation in particular beyond the project duration. The recommendation service is going to be offered prior to the resources’ reservation process, aiming to overcome related entry-point barriers. In this way, the users will be intelligently navigated through a user-friendly interface towards the examination and recommendation of the potential resources that can host their experiment, based on their needs, while significantly improve the end user experience - especially for newcomers - and reduce the learning curve for using the Fed4FIRE+ services.

Through the declaration of a set of requirements and preferences, the end-user will receive suggestions for reserving resources in the candidate testbeds that fulfil these requirements. Following, a set of filters will be designed and applied, where the end-user will be able to eliminate the suggestions for the suitable resources, upon the provision of requirements for their experiments (e.g., need for specific wireless nodes, need for IoT nodes, need for computational power). The recommendation service is going to be made available through a web portal. This portal can complement the existing “Getting Started” process in particular for the new experimenters/ or non-experts. The recommendation service is going also to be interlinked with the reputation service that provides reputation values for all the involved testbeds.

The architectural approach of the recommendation service is depicted at Figure 17. The jFed software is going to be used to collect information regarding the available resources per testbed. Such information will be stored in an internal database (for the recommender) and be updated periodically. Information coming from the reputation service will be also stored per testbed. Based on the available information, a rule-based management system is going to be developed to support the production of recommendations. A set of filters will be made available in the frontend part, where the end users will be able to provide details for the type of resources that are required for its experiment. Based on these requirements, the end users will receive suggestions for reserving resources in the candidate testbeds that fulfil these requirements.

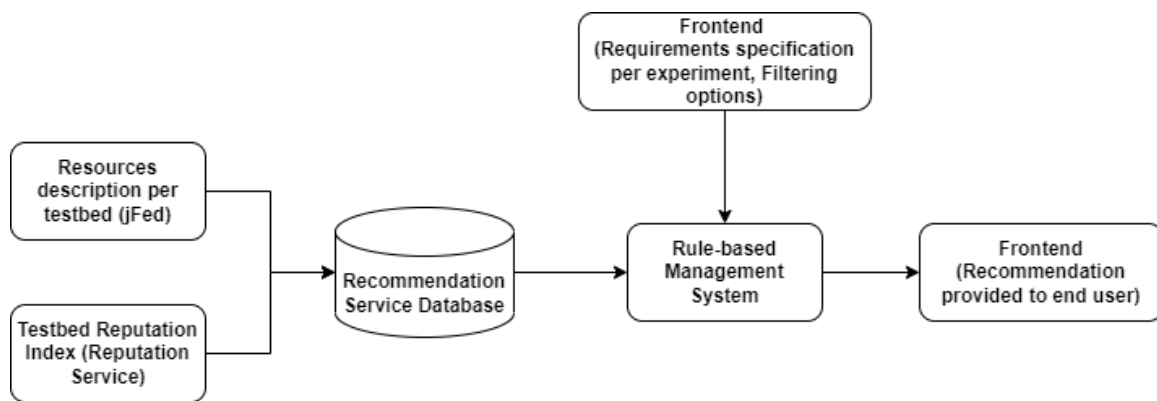


Figure 17: Resource Recommendation Service Architectural Approach

7 AUTOMATED OPENSTACK DEPLOYMENT

Because of a high user demand for an easy to set up Openstack environment, we added this requirement for the 2nd cycle. In cycle 2 we developed the necessary scripts and documentation (ESpec, <https://doc.ilabt.imec.be/ilabt/virtualwall/tutorials/openstack.html>) to deploy automatically a flexible openstack environment in the Fed4FIRE+ environment. This is based on the existing frameworks of as EnOS (<http://beyondtheclouds.github.io/>). See also Figure 6.

We got the feedback that this worked well, but after some time it was outdated because some component (ubuntu, openstack, EnOS, ...) was updated and the whole scripting did not work. So, there was a requirement to do automatic nightly testing, so we can detect problems early on.

For this, we combined our existing Federation Monitor tool (<https://fedmon.fed4fire.eu>) and added a nightly test with the Openstack ESpec including also some basic tests (starting a virtual machine and verifying it runs correctly). Figure 18 shows where one can find the test on the Federation Monitor website while Figure 19 shows the latest runs. As one can see, this is our most complex nightly test, taking up to 31 minutes. One can indeed see that an experimenter might run into problems using this ESpec. We are currently working on solving these issues.

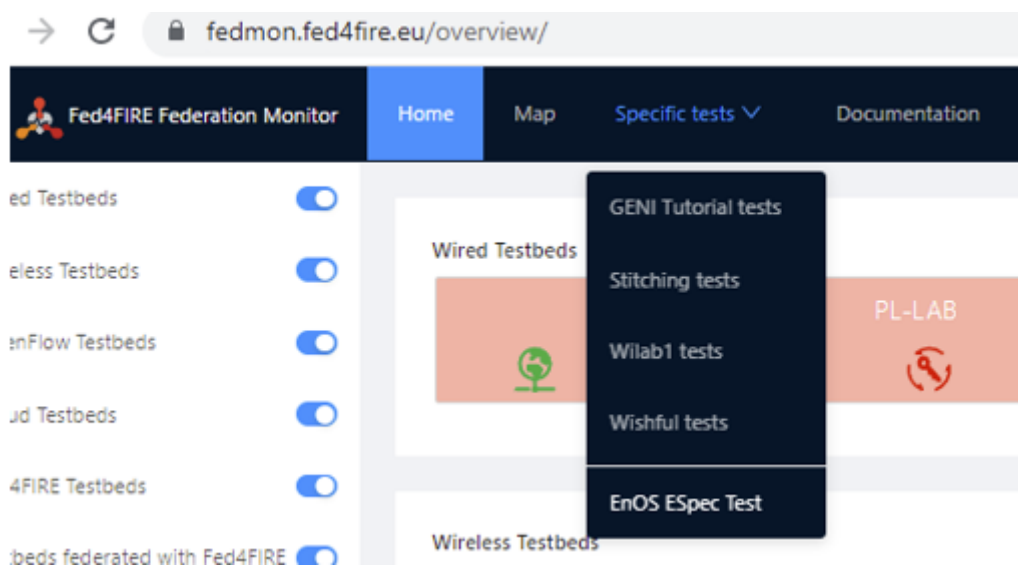


Figure 18: Specific test for the Openstack ESpec on the Federation Monitor website

D3.6: Developments for the 3rd cycle



Start	Duration	Summary	Actions
Wednesday, March 16, 2022 1:01 AM	31 minutes, 7.269 seconds	Success	Details
Tuesday, March 15, 2022 12:32 AM	2 minutes, 6.092 seconds	Failure	Details
Monday, March 14, 2022 12:32 AM	2 minutes, 8.902 seconds	Failure	Details
Sunday, March 13, 2022 12:32 AM	2 minutes, 8.862 seconds	Failure	Details
Saturday, March 12, 2022 12:32 AM	2 minutes, 6.002 seconds	Failure	Details
Friday, March 11, 2022 12:32 AM	2 minutes, 5.7 seconds	Failure	Details
Thursday, March 10, 2022 12:56 AM	26 minutes, 37.194 seconds	Success	Details
Wednesday, March 9, 2022 8:13 AM	26 minutes, 6.509 seconds	Success	Details
Wednesday, March 9, 2022 12:55 AM	25 minutes, 6.912 seconds	Failure	Details
Tuesday, March 8, 2022 12:55 AM	25 minutes, 8.159 seconds	Failure	Details
Monday, March 7, 2022 12:55 AM	25 minutes, 6.719 seconds	Failure	Details
Sunday, March 6, 2022 12:55 AM	25 minutes, 6.895 seconds	Failure	Details
Saturday, March 5, 2022 12:55 AM	25 minutes, 7.493 seconds	Failure	Details
Friday, March 4, 2022 12:55 AM	25 minutes, 7.003 seconds	Failure	Details
Thursday, March 3, 2022 12:55 AM	25 minutes, 6.875 seconds	Failure	Details
Wednesday, March 2, 2022 12:55 AM	25 minutes, 9.96 seconds	Failure	Details
Tuesday, March 1, 2022 12:55 AM	25 minutes, 6.528 seconds	Failure	Details
Monday, February 28, 2022 12:55 AM	25 minutes, 8.587 seconds	Failure	Details
Sunday, February 27, 2022 12:55 AM	25 minutes, 7.011 seconds	Failure	Details
Saturday, February 26, 2022 12:55 AM	25 minutes, 7.037 seconds	Failure	Details
Friday, February 25, 2022 12:37 AM	7 minutes, 8.204 seconds	Success	Details
Thursday, February 24, 2022 12:40 AM	10 minutes, 7.974 seconds	Success	Details
Wednesday, February 23, 2022 12:40 AM	10 minutes, 10.486 seconds	Success	Details
Tuesday, February 22, 2022 12:40 AM	10 minutes, 6.369 seconds	Success	Details
Monday, February 21, 2022 12:40 AM	10 minutes, 8.296 seconds	Success	Details

Figure 19: Latest runs of the Openstack ESPEC

8 CONCLUSIONS

This deliverable describes the developments in WP3 for cycle 3 of the project.

There are 5 tasks in the WP and the topics addressed are quite extensive:

- Task 3.1 SLA and reputation for testbed usage
- Task 3.2 Experiment-as-a-Service (EaaS), data retention and reproducibility of experiments
- Task 3.3 Federation monitoring and interconnectivity
- Task 3.4 Service orchestration and brokering
- Task 3.5 Ontologies for the federation of testbeds

The developments of the SLA and reputation services went further on the first cycle developments. More specifically, a mature version of the service was deployed and tested with the i2CAT and IRIS testbeds and the integration with jFed was updated.

For the new authority/portal, also new certificates were introduced and described in this deliverable.

For improving the reproducibility of experiments, we further refined the Experiment Specification (ESpec) which helps in reproducing experiments, including the provisioning phase, while we added a new jFed CLI 2 command line tool which supports now ESpecs and which has a workflow more resembling the jFed GUI workflow. The Experiment Orchestration tool (ExpO), meant as a lightweight orchestration tool during the experiment phase (after provisioning), was further updated.

In this cycle jupyter notebooks were studied as well and are supported on both the GPULab testbed and the Grid'5000 testbed.

For the Grid'5000 testbed a tool was developed which collects all details about an experiment (e.g. hardware description of resources used in an experiment, software versions, monitoring data). It takes a lot of time to collect all this manually (without forgetting anything), so this experiment metadata bundler tool does this now.

Another tool, called Distrinet, was developed to support large scale network experimentation by distributing network emulation on multiple nodes.

In this cycle Grid'5000 was also further federated with Fed4FIRE+ and a full lessons-learned is included in this deliverable to help other testbeds in federating.

The central broker for resources has been integrated with the portal to help in testbed/resource selection for new experimenters.

Another tool (Resource Recommendation Service RRS) was started to serve a similar goal: to help experimenters select the most appropriate resources (across the available FED4FIRE+ testbeds) for realising an experiment, while exploiting its capabilities at full potential through the use of the federation as a whole (and not in a fragmented manner through the use of specific testbeds). This is work in progress that will be finished by the end of the project.

Finally, we have added a nightly test for the ESpec setting up an automated open stack deployment. This is a quite complex setup (taking up to 30 minutes to fully deploy) and seems to fail a lot. The nightly test will help us improve this ESpec and the testbeds.